Massachusetts
Institute
of Technology

Project MAC
Progress Report VIII
July 1970 to
July 1971

The work reported here was carried out within Project MAC, an M.I.T. interdepartmental laboratory. Support was provided by:

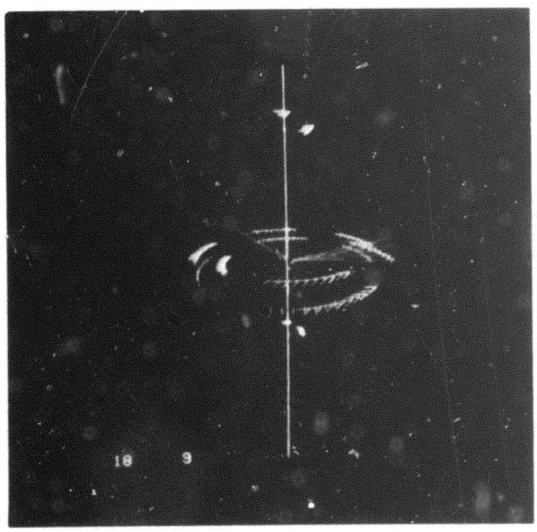
The Advanced Research Projects Agency of the Department of Defense, under Office of Naval Research Contracts N00014-70-A-0362-0001, -0002, and Defense Supply Service Contract DAHC15 69 C-0347;

The Office of Naval Research under Contract N00014-69-A-0276-0002;

The National Aeronautics and Space Administration under Contracts NGR 22-009-393 and NAS 12-2093;

The National Science Foundation, under Contracts GJ-432 and GJ-1049.

The support for some of this work came from the M.I.T. Departments and laboratories that participate in Project MAC and whose research programs are, in turn, sponsored by Government and private agencies.



The cover and above pictures show displays of the states of a three-dimensional cellular automaton. (A cellular automaton is a type of parallel processing computer composed of an array of identical, simple processing units called cells.) In these simulations of an array of three-state cells, state 2 is represented by an incomplete triangle, state 1 by a 1 and state 0 by the absence of a symbol. Such generalized tessellation automata have been studied at Project MAC and are described on page 9.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Massachusetts Institute of Technology Project MAC	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP None

3. REPORT TITLE

Project MAC Progress Report VIII July 1970 to July 1971

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Annual Progress

5. AUTHOR(S) (Last name, first name, initial)

Collection of reports from Project MAC participants
Profs. J. C. R. Licklider and Edward Fredkin

6. REPORT DATE 1 July 1971	7a. TOTAL NO. OF PAGES 235	7b. NO. OF REFS (In Text)
8a. CONTRACT OR GRANT NO. N00014-70-A-0362-0001, -0002; DAHC 69 C-0347; N00014-69-A-0276-0002; NGR 22-009-393 and NAS 12-2093; GJ-432 and GJ-1049	9a. ORIGINATOR'S REPORT NUMBER(S) MAC Progress Report VIII 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) --	

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT

The broad goal of Project MAC is experimental investigation of new ways in which on-line use of computers can aid people in their individual work, whether research, engineering design, management, or education.

This is the eighth annual Progress Report summarizing the research carried out under the sponsorship of Project MAC. Details of this research may be found in the publications listed at the end of each section and in Appendix A.

14. KEY WORDS		
On-Line Computers	Cellular Automata	Education
Multiple-Access Computers	Time-Sharing	Dynamic Modeling
Real-Time Computers	Information Systems	Programming Languages
Computer Networks	Artificial Intelligence	Computation Structures
Interactive Management	Machine-Aided Cognition	Automata Theory
	Graphics	Implicit Computation

DD FORM 1 NOV 68 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

PROJECT MAC
PROGRESS REPORT VIII
JULY 1970 to
JULY 1971

ARTIFICIAL INTELLIGENCE

AUTOMATA THEORY

CELLULAR AUTOMATA

COMPUTATION STRUCTURES

COMPUTER SYSTEMS RESEARCH

DYNAMIC MODELING, GRAPHICS AND NETWORKS

EDUCATION

IMPLICIT COMPUTATION

INTERACTIVE MANAGEMENT SYSTEMS

MATHLAB

PROGRAMMING LANGUAGES

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

PROJECT MAC PERSONNEL

JULY 1970 to JULY 1971

Administration

Prof. J. C. R. Licklider	Director (to June 1971)
Prof. E. Fredkin	Director
Prof. M. M. Jones	Assistant Director (to June 1971)
D. C. Scanlon	Administrative Officer
D. E. Burmaster	Assistant Director for Student Activities (to June 1971) and Business Manager (to December 1970)
G. B. Walker	Business Manager
P. Brandler	Assistant Business Manager (to September 1970)
R. J. Harman	Assistant to the Director
M. S. Draper	Administrative Assistant (to June 1971)
M. K. Hadley	Librarian (to December 1970)
B. H. Kohl	Librarian

Academic Staff

Prof. F. J. Corbató
Prof. J. B. Dennis
Prof. M. L. Dertouzos
Prof. J. J. Donovan
Prof. A. Evans, Jr.
Prof. R. M. Fano
Prof. R. R. Fenichel
Prof. E. Fredkin
Prof. G. A. Gorry, Jr.
Prof. F. C. Hennie, III
Prof. M. M. Jones
Prof. J. C. R. Licklider
Prof. C. L. Liu
Prof. W. A. Martin
Prof. A. Meyer
Prof. J. Moses
Prof. N. P. Negroponte
Prof. S. S. Patil
Prof. J. H. Saltzer
Prof. J. F. Shapiro
Prof. J. Weizenbaum

Instructors, Research Associates, Research Assistants, and Others

R. G. Abramson
V. Altman
D. Asthana
A. Bagchi
R. Barquin
R. D. Bressler
D. Brown
G. G. Bruere-Dawson
R. H. Bryan
R. Bryant
I. R. Campbell-Grant
D. D. Clark
J. Coffman
J. D. DeTreville
M. W. Dixon
G. T. Dixon
R. S. Eanes
R. Earle
M. Edelberg
R. J. Fateman
R. J. Fleischer
J. Fosseen
P. J. Fox
F. Furtek
R. C. Goldstein
A. Gonzales
L. I. Goodman
I. G. Greif
F. E. Guertin
M. Hack
M. Hammer
J. F. Haverty
I. T. Hawryszkiewycz
P. G. Hebalkar
D. A. Henderson, Jr.
G. Holt
P. M. Hutchins
J. Johnson
R. Johnston
M. E. Kaliski
J. Kaplan
D. J. Kfoury
P. A. King
W. J. Klos
D. König
B. Lester
J. P. Linderman
M. P. Lum
N. A. Lynch
C. W. Lynn
S. E. Madnick
R. Mandl
F. Manning
M. J. Marcus
S. P. Mason
D. T. McDonald
W. C. Michels
P. L. Miller
R. N. Moll
A. R. Monroe-Davies
M. L. Morgenstern
S. Murthy
B. G. Ong
H. F. Okrent
P. Olson
R. C. Owens, Jr.
G. Pfister
K. T. Pogran
D. H. Porges
C. Ramchandani
L. J. Rotenberg
J. E. Rumbaugh
R. R. Schell
M. D. Schroeder
J. I. Seiferas
A. Sekino
W. G. Shaw
D. G. Sitler
J. R. Sloan
B. J. Smith
J. R. Stinger
R. H. Thomas
H. M. Toong
L. E. Travis

Instructors, Research Associates, Research Assistants, and Others (cont.)

B. J. Vilfan
C. M. Vogt
W. C. Walker
P. S. Wang
S. A. Ward
A. S. Weinberg
T. A. Welch
C. Ying
D. Yun

Undergraduate Students

W. F. Bauer
A. J. Baum
P. B. Bishop
R. M. Berman
E. H. Black
P. G. Bras
J. A. Brenfleck
D. Bricklin
H. Brodie
R. L. Brooks
M. S. Broos
K. M. Brown
R. H. Brown
B. Carlson
A. Y. Chan
D. J. Chang
D. M. Christie
S. S. Cohen
J. R. Cone
R. G. Curley
S. E. Cutler
B. K. Daniels
R. Davis
R. M. Elkin
R. Frankston
R. A. Freedman
D. E. Geer
M. R. Genesereth
R. S. Goldhor
P. A. Green
R. A. Guida
P. H. Guldberg
R. H. Gumpertz
J. H. Harris
C. A. Hatvany
B. Hubbard
P. W. Hughett
W. F. Hui
J. E. Jagodnik
E. Kant
P. A. Karger
R. M. Katz
C. A. Kessel
H. J. Kim
R. N. King
E. Kohn
N. V. Kohn
D. M. Krackhardt
P. B. Kurnik
R. S. Lamson
P. J. Leach
P. D. Lebling
C. K. Leung
J. C. Lind
M. Liu
W. S. Mark
J. R. McCauley
D. Misunas
S. Morrow
S. G. Morton
W. Y. Ng
P. A. Pangaro
G. Pavel
R. Pincus
R. L. Prakken
J. Quimby
D. P. Reed
J. L. Reuss
K. Rhoades

Undergraduate Students (cont.)

M. A. Rondio
E. C. Rosen
J. L. Rosenberg
L. M. Rubin
N. D. Ryan
S. Saunders
H. J. Siegel
R. P. Silberstein
N. Singer
A. M. Solish
J. Stern
S. M. Stoney
J. M. Strayhorn
C. D. Tavares
W. H. Thrasher
E. Tsiang
H. Tucker
L. E. Widman
B. J. Zak
R. E. Zippel

DSR Staff

M. J. Ablowitz
G. J. Bailin
E. R. Banks
A. K. Bhushan
G. R. S. Bingham
P. Brandler
A. L. Brown
D. E. Burmaster
B. Byer
R. H. Campbell
H. O. Capps
M. A. Cohen
D. G. Cressey
R. C. Daley
S. D. Dunten
A. C. England
R. J. Feiertag
J. A. Friel
S. W. Galley
R. L. Gardner
C. C. Garman
R. P. Goldberg
J. P. Golden
P. M. Gunkel
R. J. Harman
J. P. Jarvis, III
R. K. Kanodia
B. H. Kohl
M. Lenot
R. F. Mabee
K. J. Martin
R. M. Metcalfe
E. W. Meyer, Jr.
J. C. Michener
N. I. Morris
S. E. Niles
M. A. Padlipsky
S. G. Peltan
W. W. Plummer
C. L. Reeve
L. P. Rothschild
D. C. Scanlon
R. Schroeppel
T. P. Skinner
M. J. Spier
A. J. Strnad
J. Taggart
R. C. Thurber, Jr.
A. Vezza
V. L. Voydock
G. B. Walker
M. B. Weaver
S. H. Webber
D. M. Wells

Support Staff

M. C. Amyot
M. E. Baker
V. M. Berardinelli
M. A. Bizot
M. F. Brescia
O. D. Carey
L. S. Cavallaro
N. Chen
M. T. Cheney
S. J. Cohn
M. J. Connell
J. Considine
S. Daise
L. K. Denison
C. P. Doyle
C. Falls
L. L. Gammell
A. M. Garrity
R. E. Golden, III
D. Goldthrope
M. J. Grano
M. K. Hadley
J. A. Haley
L. J. Haron
A. J. Hicks
R. F. Hill
M. A. Hoer
D. L. Jones
D. Kontrimus
E. T. Moore
B. A. Morneault
E. F. Nangle
L. G. Pantalone
K. W. Pierce
J. E. Pinella
R. Pinsley
R. Queens
E. M. Roderick
A. Rubin
T. H. Seymore
K. K. Simpson
A. H. Speare
J. Stavrinos
M. K. Stephens
J. E. Tamayo
A. G. Testa
E. B. Ulman
M. W. Webber
L. E. Yaple
F. L. Yost
K. Young

Guests

H. Adler
Prof. J. Berger
P. Eisenberg
Prof. J. I. Elkind
Prof. A. Fleisher
Prof. G. Iazeolla

ADMINISTRATION

Academic Staff

Prof. J. C. R. Licklider	Director (to June 1971)
Prof. E. Fredkin	Director
Prof. M. M. Jones	Assistant Director (to June 1971)

Administrative Staff

D. C. Scanlon	Administrative Officer
D. E. Burmaster	Assistant Director for Student Activities (to June 1971) and Business Manager (to December 1970)
G. B. Walker	Business Manager
P. Brandler	Assistant Business Manager (to September 1970)
R. J. Harman	Assistant to the Director
M. S. Draper	Administrative Assistant (to June 1971)
M. K. Hadley	Librarian (to December 1970)
B. H. Kohl	Librarian

Research Staff

P. M. Gunkel

Research Assistant

F. Manning

Support Staff

M. C. Amyot
L. S. Cavallaro
M. J. Connell
J. Considine
L. K. Denison
L. L. Gammell
J. E. Goss
N. P. Greeley
D. Kontrimus
E. T. Moore
L. G. Patanlone
K. W. Pierce
R. Pinsley
E. M. Roderick
K. K. Simpson
A. H. Speare
J. Stavrinos
E. B. Ulman

PREFACE

Project MAC was begun as an interdepartmental laboratory at the Massachusetts Institute of Technology in early 1963. The initial research and development goals were concerned with Multiple-Access Computer systems, Machine-Aided Cognition, and, in general, the interaction between Men And Computers. The name "MAC" is an acronym for each of these goals.

In the year ending June, 1971, there were 320 persons associated with MAC. They included: 21 faculty members mainly from the Departments of Electrical Engineering and Mathematics and from the Alfred P. Sloan School of Management; 105 staff members (DSR Staff and Support Staff), 182 students (Undergraduate and Graduate) and 12 Guests.

Early in its history, MAC conducted extensive experimentation with and development of the Compatible Time-Sharing System (CTSS), an early large-scale, multiple-access computer system. More recently we have continued our research on the MULTICS system, which came into operation 2 years ago. MULTICS is a conceptually advanced multiple-access system that is capable of straightforward and smooth expansion into an extremely large and capable facility.

The second of MAC's original objectives, machine-aided cognition, has recently made very significant progress. We feel that recent MAC/AI research represents an enormous conceptual advance. In December, 1970 the Artificial Intelligence group became an independent MIT laboratory; Professors Marvin Minsky and Seymour Papert are Co-directors. Important and useful collaboration between MAC and the AI Laboratory is continuing.

In May, 1971, Professor J. C. R. Licklider stepped-down from the Directorship of MAC to devote full time to his own research specialties - Dynamic Modeling, Computer Graphics, and Computer Networks - and Professor Edward Fredkin assumed the Directorship. Miss Dorothea Scanlon continued as Administrative Officer, and Mr. Gary Walker remained as Business Manager.

In anticipation of a major research thrust in a new direction, MAC has consolidated and strengthened various groups. Educational Applications; MacAIMS; Programming Linguistics/ Extensible Languages; and Programming Linguistics/ Formal Systems have been terminated as separate groups. A policy of more decentralized control by the group leaders has been instituted.

Although the specific goals of MAC for the next few years are now the subject of much thought and discussion, an emerging consensus seems to be that we are interested in the problems of imbedding knowledge in the computer and in enabling that knowledgeable system to play a key role in generating programs and other forms of solutions to problems. We feel that, armed with knowledge, a system will be able to better communicate with its users. We give this field the name "Automatic

Programming".

This progress report outlines the research carried out in the year ending June, 1971. The report is subdivided into 11 sections corresponding to the research groups in Project MAC. The technical reports and memoranda of Project MAC are listed in Appendix A, and references to the external publications resulting from the research appear in the bibliographies at the end of each section.

During the past year, the core program of Project MAC and the Artificial Intelligence Group were supported, as heretofore, by the Information Processing Techniques Directorate of the Advanced Research Projects Agency (ARPA). Individual projects were funded by several other agencies: research in extensible languages, National Aeronautics and Space Administration; interactive problem-solving and decision-making, Office of Naval Research; dynamic modeling, Behavioral Sciences Directorate of ARPA; programming generality, National Science Foundation.

Edward Fredkin

Cambridge, Massachusetts

AUTOMATA THEORY

Prof. F. C. Hennie, III

Academic Staff

Prof. C. L. Liu
Prof. A. Meyer

Instructors, Research Associates, Research Assistants and Others

A. Bagchi
D. Brown
G. G. Bruere-Dawson
M. Edelberg
M. M. Hammer
P. M. Hutchins
D. J. Kfoury
N. A. Lynch
R. Mandl
R. N. Moll
B. G. Ong
B. J. Smith
B. J. Vilfan
C. Ying

Support Staff

M. E. Baker
V. M. Berardinelli
S. J. Cohn

II. AUTOMATA THEORY

Abstract complexity theory, which has been a central topic of research in the Automata Theory Group in the past, has become a reasonably developed chapter in the theory of computability, with contributions from nearly three dozen authors in the U.S. and the Soviet Union. As reported below, some further work in this area was carried out during this last year, and two doctoral theses are still in progress. However, the basic phenomena associated with the classification of computations according to their time and space requirements are now rather well understood, and further refinements in the abstract theory are likely to be of diminishing interest to the computer scientist. Major interest within the group has now shifted toward combinatorial and statistical analyses of a variety of algorithms commonly arising in computation. The goals of work in this area are to develop methods for designing good algorithms for problems of practical interest, and to devise techniques for verifying the optimality of algorithms. The work described below on matrix multiplication, polynomial evaluation, and sorting represents the beginnings of this more practical approach to the study of algorithms.

A. Abstract Complexity Theory

One of the basic theorems about computable functions is that, for every computable function t, there exists a zero-one valued computable function c that takes more than time t to evaluate. More precisely, any program that evaluates c requires at least t(x) steps to compute c(x) for all but finitely many values of x.

In order to appreciate the significance of such a theorem, one needs additional information (not provided by the usual proof) about how many values of the function c are easy to compute. It might be the case that the functions that are difficult from the point of view of complexity theory -- i.e., functions that are time-consuming to compute on the average -- are actually easy for all small arguments, say all arguments less than 10^{100}. In fact, any zero-one valued function can be computed rapidly for any given finite set of arguments by simply storing the pertinent values of the function in a table. A genuinely complex function should have the property that any program that computes it can run rapidly on only as many inputs as can be stored in a table whose size equals that of the program. Such functions are constructed and studied in a paper by Prof. A. Meyer (jointly with E. M. McCreight).

Properties of program size are considered in several further papers written this year. One of the motivations for the study of program size has been to provide a quantitative understanding of the relative convenience of different programming languages by comparing the sizes of the programs needed to implement the same computation in different languages. A fairly general theorem recently proved by Prof. Meyer shows that a slight increase in the set of instructions of certain kinds of programming languages can lead to enormous economies in program size. A related study of formal grammars by Meyer (jointly with Prof.

M. J. Fischer) derives quantitative bounds on the improvement in simplicity of definition that can be achieved by using powerful grammars such as context-sensitive or context-free grammars to define simple sets such as regular or even finite sets.

B. Algorithms on Graphs

The results on matrix multiplication and transitive closure of graphs mentioned in last year's report have been strengthened. Robert Mandl has shown that the time required to find the transitive closure of a directed n-node graph is within a constant factor of that required to multiply two nXn Boolean matrices. When coupled with Meyer and Fischer's observation that fast algebraic methods for multiplying real matrices can be modified to apply to Boolean matrices, this result yields the best transitive closure algorithm known to date.

Our hope that a graph-theoretic approach to Boolean matrix multiplication might enable us to generalize fast matrix multiplication techniques has not yet been fulfilled, but we continue to believe that this approach is promising.

C. Polynomial Evaluation

The evaluation of rational functions by sequences of algebraic operations represents one of the few areas where techniques have been developed for establishing the optimality of algorithms. Larry Stockmeyer, together with Professors Fischer, Meyer and M. S. Paterson, has derived a lower bound of $\sqrt{n}$ on the number of multiplications required to evaluate any degree n polynomial with rational coefficients, and has shown that this lower bound is nearly achievable.

D. Sorting

B. J. Smith has been investigating sorting networks composed of two-input, two-output comparators. Since each comparator can be modeled as a three-state finite-state machine, the sorting network as a whole can also be viewed as a finite-state machine. When implemented in hardware, such networks can be used as high-speed sorters or message-switching devices. Alternatively, a sorting network can be realized by a computer program that is naturally suited to parallel evaluation.

Smith has been studying the minimum number of comparators required to construct an n-input, n-output sorting network. He has discovered that a network of comparators actually sorts if and only if the network has

$$\sum_{i=0}^{n} i!\, S(n,i)$$

states that are reachable from the starting state, where $S(n,i)$ is a Stirling number of the second kind. Furthermore, he has determined that no two distinct reachable states in the network are equivalent. These results suggest that a knowledge

of the number of internal network configurations that cannot result from any network input may yield bounds on the number of states "wasted" in building a network and in this way yield bounds on the number of comparators required.

E. Papers

During the year, several members of the group have prepared papers for forthcoming meetings and journals.

For the Twelfth Annual Switching and Automata Theory Symposium (October 1971):

1) Fischer, M. J. and A. R. Meyer, "Boolean Matrix Multiplication and Transitive Closure".

2) Meyer, A. R. and M. J. Fischer, "Economy of Description by Automata, Grammars, and Formal Systems".

For the International Symposium on the Theory of Machines and Computations (August 1971):

Meyer, A. R. and E. M. McCreight, "Computationally Complex and Pseudo-Random Zero-One Valued Functions".

Accepted by the Journal of Symbolic Logic:

Meyer, A. R. and P. C. Fischer, "Computational Speed-Up by Effective Operators".

Accepted by Zeit. f. Math. Log. und Grund. der Math.:

Meyer, A. R. and D. M. Ritchie, "A Classification of the Recursive Functions".

Publication 1970-1971

Ying, C. and A. K. Susskind,* "Building Blocks and Synthesis Techniques for the Realization of M-ary Combinational Switching Functions", Proceedings of Symposium on Theory and Applications of Multiple-Valued Logic Design, State University of New York at Buffalo, May 1971.

* Non-MAC author.

CELLULAR AUTOMATA

Prof. E. Fredkin

DSR Staff

E. R. Banks

Undergraduate Students

W. S. Mark

III. CELLULAR AUTOMATA

A Ph.D. thesis by Roger Banks describes an investigation of a class of parallel processing computers called Cellular Automata. A cellular automaton consists of an array of simple, identical finite-state machines called cells. Each cell communicates with only its immediately surrounding cells.

The chief results of the thesis include showing that a two-dimensional array of two-state cells, each of which communicates with its four-edge neighbors, can perform any (computable) computation, i.e., it can simulate a universal Turing machine.

A configuration is a specification of the states of all the cells in some area of the iterative array. Another result described in the thesis, is the existence of a self-reproducing configuration in an array of four-state cells with each cell communicating with its four-edge neighbors. This was a reduction of four states from the previously known eight-state case.

Further work by Banks and more recently by William Mark has concerned the development of a programming system for the simulation and display of very general cellular automata in one, two and three dimensions with various neighborhoods, transition rules, numbers of states, etc.

Publication 1970-1971

Banks, Edwin R., "Information Processing and Transmission in Cellular Automata", Ph.D. Thesis, Dept. of Mechanical Engineering, January 1971, also MAC TR-81, AD 717-951.

COMPUTATION STRUCTURES

Prof. J. B. Dennis

Academic Staff

Prof. R. M. Fano

Prof. S. S. Patil

Instructors, Research Associates, Research Assistants and Others

I. R. Campbell-Grant
J. Coffman
J. Fosseen
P. J. Fox
F. Furtek
I. G. Grief
M. Hack
I. T. Hawryszkiewycz
P. G. Hebalkar
B. Lester
J. P. Linderman
M. J. Marcus
C. Ramchandani
L. J. Rotenberg
J. E. Rumbaugh
D. G.Sitler
W. C. Walker

Undergraduate Students

H. J. Kim

DSR Staff

W. W. Plummer

Support Staff

B. A. Morneault

A. Rubin

IV. COMPUTATION STRUCTURES

A. Introduction

The Computation Structures Group is concerned with the study and analysis of fundamental issues arising in the design and construction of general-purpose computer systems. The research encompasses hardware and software aspects of computer systems, and much of the work has contributed toward establishing a common conceptual basis for both aspects. The accomplishments of the past year are principally in two areas: One is the theoretical study of Petri nets as a model for asynchronous systems of interacting parts, and the realization of Petri nets in the form of speed-independent modular switching systems. The goal of this work is to build a sound theory to serve as the basis of a new methodology for the design of asynchronous digital systems. The second area is the evolution of a base program language. This effort is expected to lead to a practical formal definition scheme for source programming languages and will provide a sound basis for the functional design of advanced computer systems.

B. Petri Nets

As reported last year, we have found Petri nets to be an elegant formalism for representation of concurrency in processes and for studying asynchronous systems. Petri nets stand out in relation to other schemes because of the preciseness and ease with which they can express parallel actions, resolution of conflicts, and interaction among processes. Moreover, they have the simple structure that is essential for analytic study. Simple as they are in their structure, study of the general class of Petri nets is difficult because of the variety of situations they can represent. A study of subclasses of Petri nets which represent simpler situations is a necessary step toward understanding the general class of Petri nets, and such study has been an important objective of the group in the past year. We have identified several subclasses of interest and have found useful results about them. Before discussing these results, we present a brief introduction to Petri nets and the subclasses of interest.

A Petri net [1,2] is a directed graph which can have two types of nodes, namely transitions and places, where the directed arcs can connect only transitions to places and places to transitions (Fig. 1.). In drawing the graph, places are represented by circles and the transitions by bars. The places from which arcs are incident on a transition are called input places of the transition, terminate are called the output places of the transition. Each place can have markers (sometimes called tokens) in them. A transition having markers in all of its input places is said to be enabled. Only enabled transitions can fire; in the act of firing, the transition picks one marker from each of its input places and puts a marker in each of its output places. The marking distribution in the net changes as transitions fire, and each new marking distribution makes firing of other transitions possible. With regard to the firing of transitions, an important situation is when

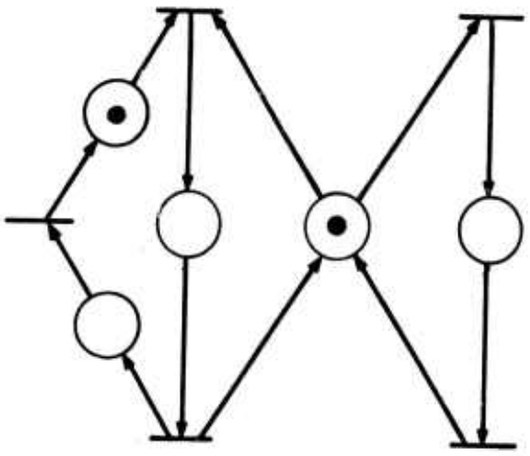


FIG. 1. A PETRI NET.

transitions share some input places. When two transitions which have a common input place are both enabled but the common input place has only one marker, the transitions are said to be in conflict because the firing of any one of the transitions disables the other. A net is said to be safe if no place in it will ever have more than one marker at a time. A net is said to be live if at no time in the operation of the net will any transition be ruled out as a transition that may fire some time in the future. Conflict, safety, and liveness in a net depend on the initial marking distribution. There are, however, some structural restrictions which can guarantee some of these properties. By structural restrictions, we mean restrictions with regard to the arrangements of transitions and places such as the restriction that transitions not have input places in common. The restrictions we use below to define subclasses of Petri nets are purely syntactic as they define local constraints on the arrangements of transitions and places. The subclasses are:

1) State Machines (SM)
2) Marked Graphs (MG)
3) Free Choice Petri Nets (FC)
4) Simple Petri Nets (SN)

The restrictions that define these subclasses are given below. The Petri nets without any restrictions will be referred to as general Petri nets to emphasize this fact. The following text should be read together with Figures 2 and 3. Figure 2 shows what kind of local configurations of transition and places are permitted for each subclass of nets.

1. State Machines (SM) -- A state machine is a Petri net in which every transition has exactly one input place and exactly

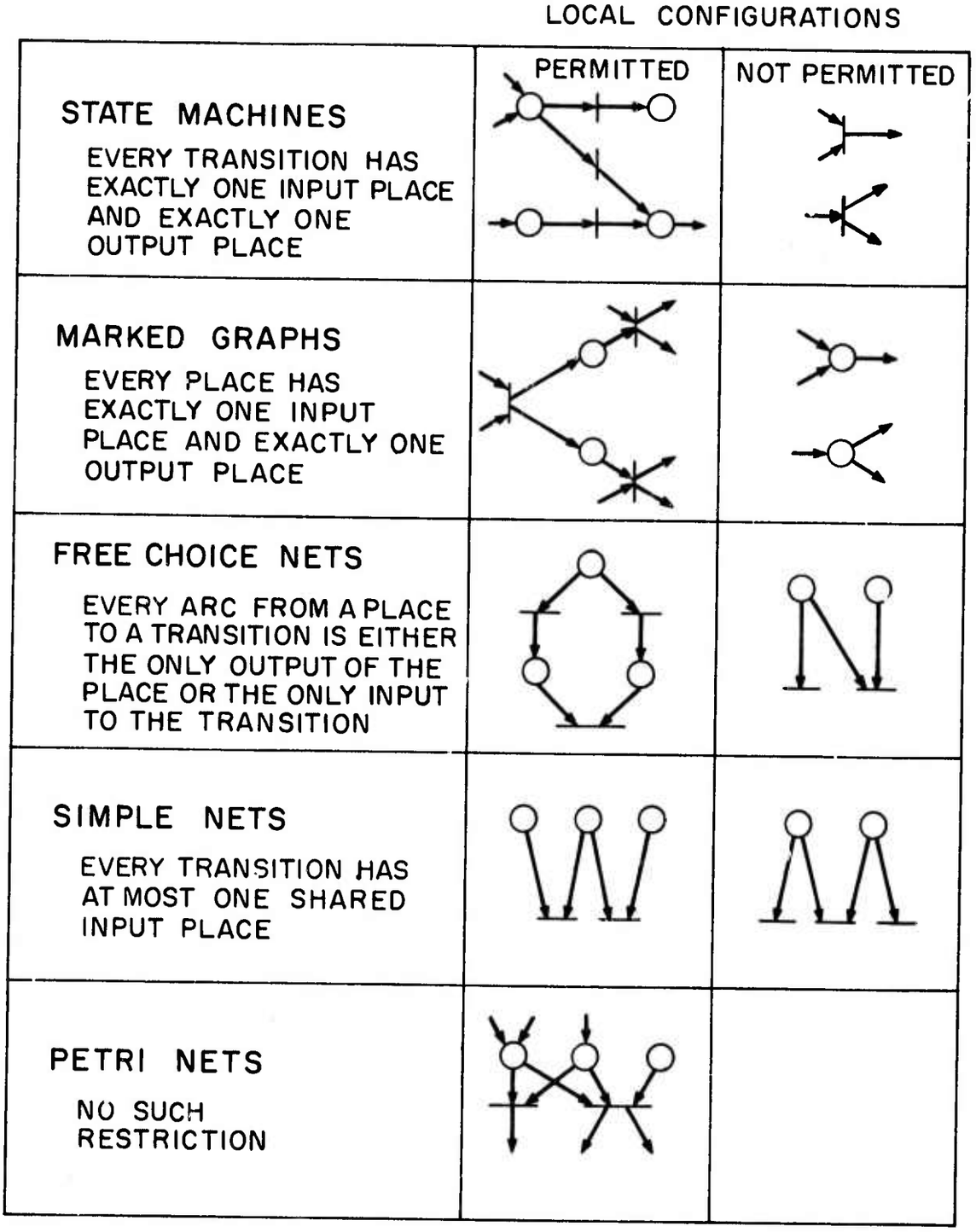


FIG. 2. THE SUBCLASSES OF PETRI NETS.

one output place. The state machines being discussed here are identical to the state machines of automata theory in their structure,(Fig. 4).

2. Marked Graphs (MG) -- A marked graph is a Petri net in which every place has exactly one input transition and exactly one output transition. Thus the restriction in this case is similar to the one for state machines but it applies to places instead of transitions. State machines have been studied extensively but the recognition of marked graphs and the study of their properties is recent. Genrich [3] started the study of marked graphs and his ideas led to a detailed study by Holt and Commoner [4]. The mathematics relating to marked graphs is fairly well understood now through these studies. In our previous report we showed a direct relationship between the elementary asynchronous modular control structures developed by us and the marked graphs. The study provided a simple way for obtaining hardware structures that mimic marked graphs, and also a method for determining if a control structure is free of any hangups. This year the study has been carried further to include a broader class of nets called free choice nets. The free choice nets and results relating to them are described below.

3. Free Choice Nets -- A Petri net in which every arc from a place to a transition is either the only output of the place or the only input to the transition is said to be a free choice Petri net. This condition on Petri nets is the same as requiring that when an input place is shared by some transitions, those transitions have no input places other than the one which is common to them. Thus when a marker arrives in the shared place, all of the transitions which share that place are enabled, and one of them may be freely chosen to fire. When the movement of a marker is regarded as flow of control, the situation just described represents a free choice with regard to where control flows from the shared place -- thus the name free choice nets. Free choice nets include both the state machines and the marked graphs.

A free choice Petri net can be used to represent the flow of control in a program as shown in Fig. 5. In this figure, the shared place x together with transitions T and F represent a decision element -- the *if* statement in the program. The direction in which control flows from place x is not arbitrary -- it conforms to the outcome of evaluating the predicate associated with the *if* statement. To the net considered alone the decision about the direction of flow is external to it because it is based on information outside the net; the information flows into the net by way of the interpretation which associates a certain *if* statement with the free choice transitions in the net. In the study of Petri nets and also in the study of computation schemata, it is important to distinguish what information is a part of the net and what is external to it.

Some important results about free choice nets have been found recently by Commoner of Applied Data Research and Hack of the

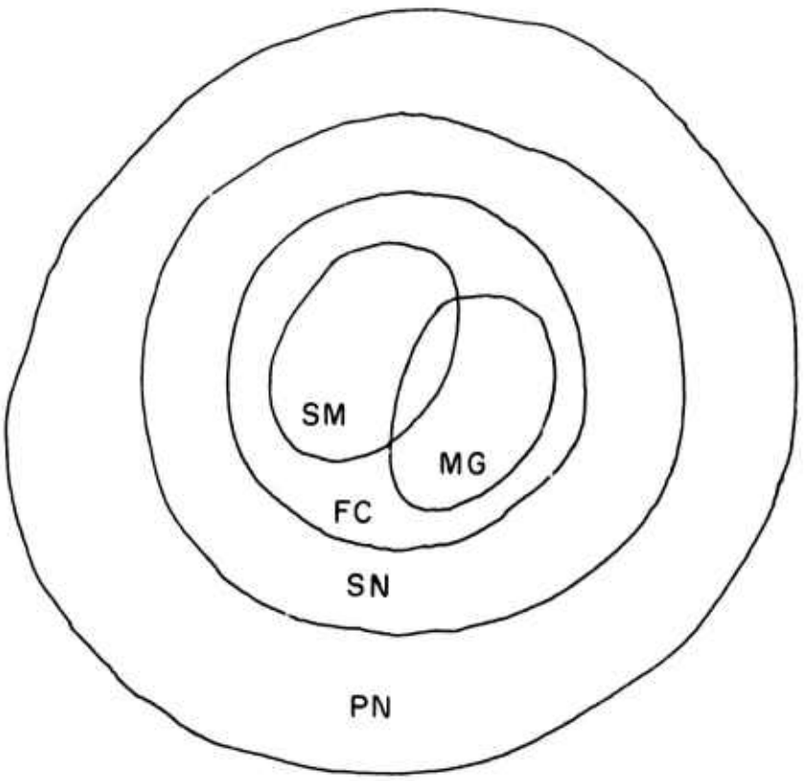
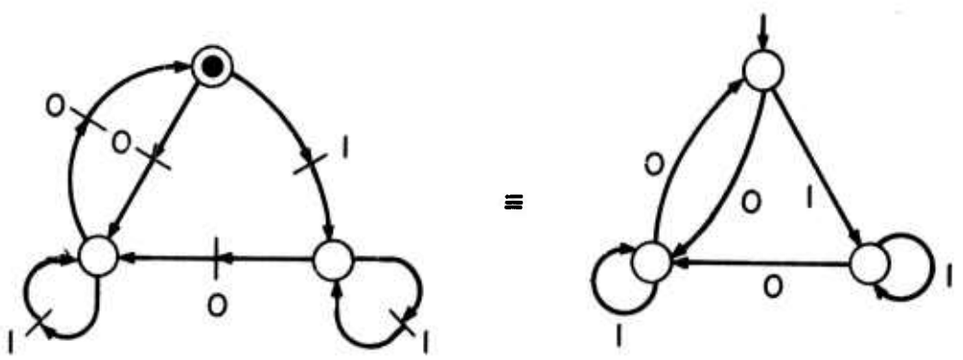


FIG. 3. THE INCLUSION RELATIONSHIP AMONG THE SUBCLASSES OF PETRI NETS.



STATE MACHINE AS A PETRI NET

STATE MACHINE AS A STATE DIAGRAM

FIG. 4. STATE MACHINES.

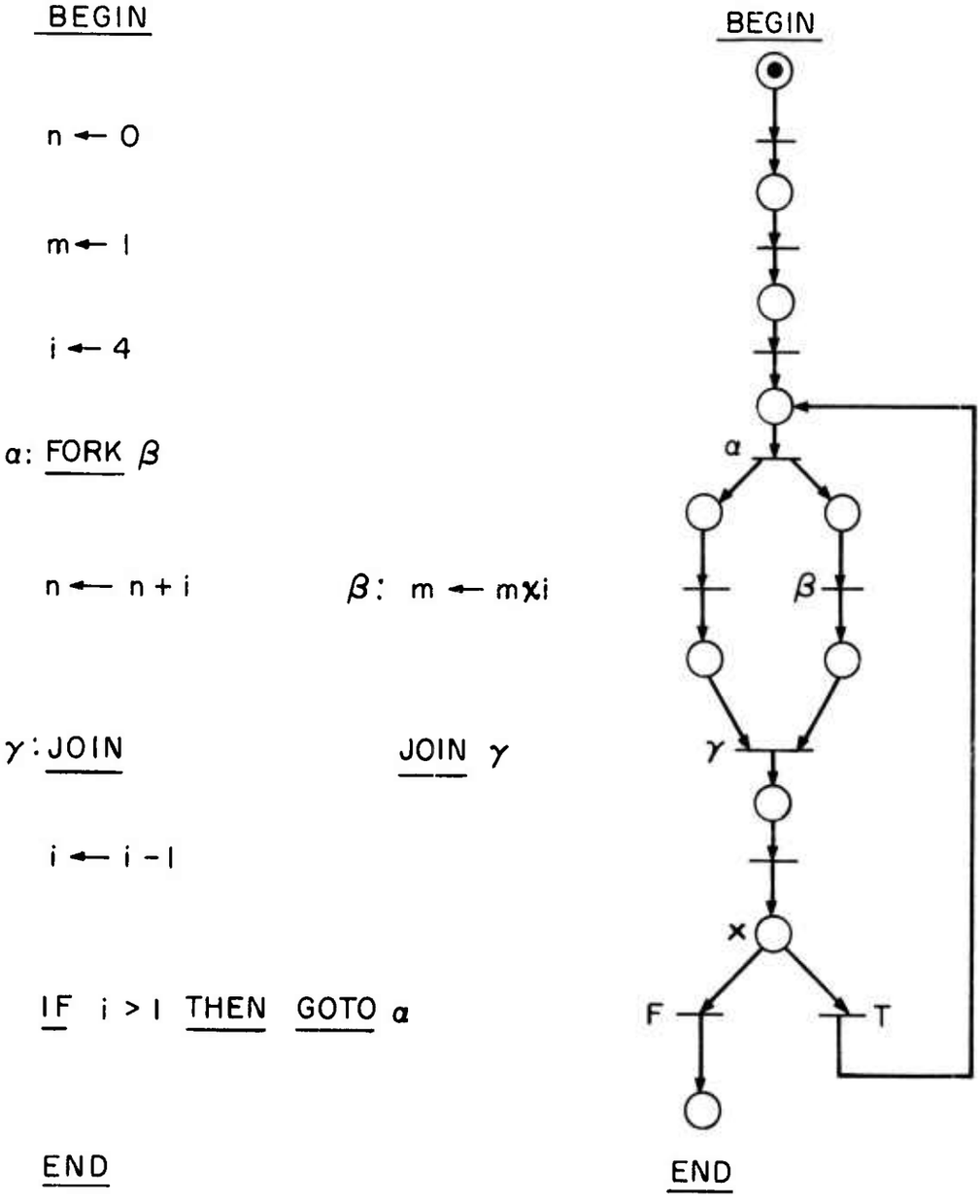


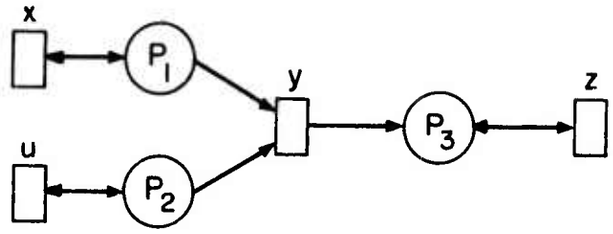
FIG. 5. FLOW OF CONTROL IN A PROGRAM.

Computation Structures Group. Commoner has found necessary and sufficient conditions for liveness and safety of a free choice net, and Hack has found conditions for the existence of a live and safe marking for a net. A live net is one in which the activity can continue indefinitely without any hangup. Hangup is a condition in which a part of the net enters into a state of inactivity from which it cannot recover. In our common experience a hangup for a machine is an unfortunate state in which its activity subsides and it fails to respond to stimulation because of some hopeless jam inside it. Safety on the other hand means that no more than one token will be in any place at any time. This is important where the places represent objects that cannot hold more than one of the things represented by the tokens. When places represent registers in a digital computer, safety means that a new piece of data will not be placed in a register until the previous one has been used up. In that way mixup of data can be avoided. Hack's work thus provides a way to determine if an uninterpreted parallel program which can be expressed as a flow diagram has a starting condition for which it will continue to operate without any hangups or mixups.

4. Simple Petri Nets -- A Petri net in which no more than one input place of any transition is a shared input place is called a simple Petri net; a transition in a simple Petri net may have any number of input places but at most one of those places may be an input place of some other transition. The class of simple Petri nets properly contains the free choice nets. There are situations which can be represented by simple Petri nets but not by free choice nets. Figure 6 shows such a situation which arises in representing flow of control in coordinating processes. An important aspect of simple nets is that they are able to represent interprocess coordination such as implemented by Dijkstra's semaphore primitives. A study of simple Petri nets has led to an understanding of the limitation and capabilities of the semaphore primitives. Details of this study are presented in the next section.

5. General Petri Nets -- The class of Petri nets without any of the restrictions is called general Petri nets. There are many Petri nets in the class of general Petri nets for which there are no equivalent nets in the subclasses defined. In particular, a Petri net which cannot be transformed into a simple net arises in the study discussed below.

Recent work by Patil [5] has shown some interesting facts about the semaphore primitives of Dijkstra [6] by establishing a correspondence between the flow of control in interacting processes and Petri nets. In Fig. 6, three processes coordinate their activities with the help of semaphores. The Petri net for each individual process is obtained by representing each instruction by a transition, connecting these transitions into a chain by means of places to indicate the flow of control in that process, and placing a token in the input place of a transition to indicate the present site of control. The Petri net for a collection of interacting processes is obtained by interconnecting the nets

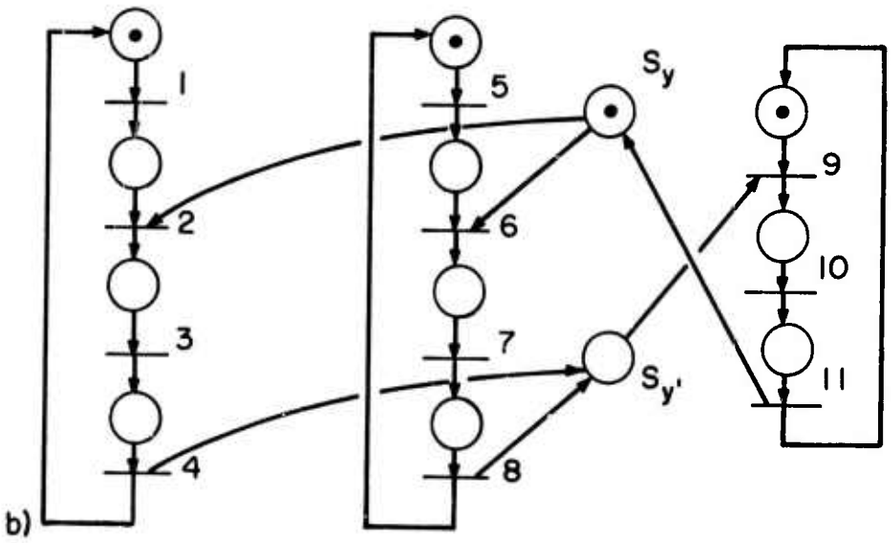


PROCESS

P_1		P_2		P_3	
1	$x \leftarrow x + x$	5	$u \leftarrow u * u$	9	$P[S_{y'}]$
2	$P[S_y]$	6	$P[S_y]$	10	$z \leftarrow z + y$
3	$y \leftarrow x$	7	$y \leftarrow u$	11	$V[S_y]$
4	$V[S_{y'}]$	8	$V[S_{y'}]$		GOTO 9
	GOTO 1		GOTO 5		

INITIALLY SEMAPHORE $S_y = 1$ AND $S_{y'} = 0$

a)



b)

FIG. 6. FLOW OF CONTROL IN PROCESSES AND THE CORRESPONDING SIMPLE PETRI NET.

for individual processes by means of places which represent the semaphores: A transition that represents an instruction P[S] is provided an input from the place that represents semaphore variable S, and each transition that represents an instruction V[S] feeds into the place representing the semaphore. The number of tokens in the place corresponding to a semaphore equals the value of the semaphore variable. Thus, corresponding to the fact that the control in a process can get past a P[S] instruction only when the process can decrement the semaphore variable S by 1, we have the phenomenon in the net that the transition corresponding to the instruction P[S] can fire only when it gets a token from the place representing the semaphore.

The above method of obtaining Petri nets for flow of control applies only to processes which do not have conditional statements. The Petri nets for such processes completely describe the flow of control. Moreover, these nets are simple Petri nets because the only transitions which can have any shared input places are the ones which correspond to the P[ ] instructions and each of these transitions has only one shared input place.

If there are any conditional instructions, they would have to be represented by two transitions, one for the outcome true and the other false, and these transitions would share the input place so that for any particular execution of the conditional instruction, only one of the transitions would fire. Which of the two transitions fires depends on the value of the predicate associated with the conditional instruction. Since this information is external to the net, the net only partially describes the flow of control in this case.

Interacting processes which do not contain any conditional statements are of particular interest to us because it would seem that the semaphore primitives would be adequate for describing their interaction, but our study has uncovered the surprising fact that the semaphore primitives are inadequate for this purpose. This fact is brought out by a study of a problem called the 2-out-of-3 problem which is discussed below.

The 2-out-of-3 problem can be explained in the framework of a message decoder. When viewed as a hardware device, the decoder has three input wires colored red, yellow and green, and three output wires called X, Y and Z. There are three different messages which can be sent to the decoder. Message X consists of signals on the red and yellow wires; message Y consists of signals on the red and green wires; and message Z consists of signals on the yellow and green wires. The decoder can be thought to have three processes inside it, one for each message. Process X waits for message X and responds on output wire X; the other processes are defined similarly. We will be concerned with the above decoder in its software form in which signals are represented through the use of semaphores; each wire is represented by a semaphore and incrementing the semaphore count by 1 corresponds to sending a signal. A

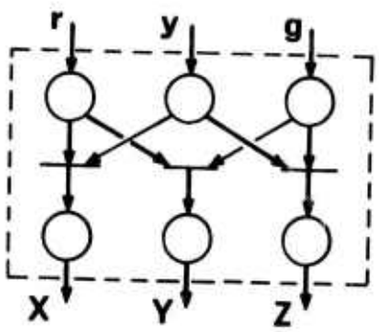


FIG.7. THE 2-OUT-OF-3 NET.

signal is accepted by decrementing the semaphore count by 1. The question is: Can the three processes which decode the messages be so coordinated by semaphore primitives that the decoder functions correctly? Since each individual process just waits for the associated message to arrive, we insist that the processes not use any conditional instructions. Therefore, instead of asking the question in the form above, we ask: Is there any finite collection of processes not using conditional instructions that can specify the operation of the decoder with the help of the semaphore primitives? The answer to this question is negative.

The reason for the negative answer is that the decoder represents a net called 2-out-of-3 net, which is not a simple Petri net, and it has been possible to show that this net cannot be transformed into an equivalent simple Petri net [5]. Thus it is clear that the semaphore primitives need the help of conditional statements to carry out coordination among processes, (Fig. 7.). It should be recalled that the very purpose of introducing the semaphore primitives was to obtain a more direct means for coordinating processes and to do away with sneaky use of conditional statements to perform coordination. With the aid of conditional statements one can implement coordination of processes by such simple-minded schemes as repeated testing of a variable until it becomes, say, 1. Such schemes can implement coordination, but the implementation is very wasteful of computer resource because there is no limit to the number of times the variable may have to be checked. The semaphore primitives rectify this defect, but they are not able to implement all coordinations by themselves. Thus the question is, whether together with conditional statements they can express all conceivable coordinations without paying the price of unbounded computation. The study has shown that the answer to this question is affirmative.

At the root of the shortcomings of the semaphore primitives is the fact that a P[ ] instruction operates on only one semaphore. Unfortunately, a generalized instruction such as $P[S_1,\ldots,S_k]$, which simultaneously operates on semaphores $S_1, \ldots, S_k$, cannot be always expanded into a sequence of instructions $P[S_1], \ldots, P[S_k]$. But the generalized instruction can be expanded in terms of $P[S_1, S_2]$ instructions each of which operates on two semaphores. Even though $P[S_1, S_2]$ is adequate, one may wish to allow more arguments in instructions for the sake of efficiency.

C. Asynchronous Speed-Independent Circuits

A digital system is often built as two interconnected parts -- a data flow structure containing registers, functional operators and data paths, and a control structure that generates signals that initiate actions by operators in the data flow structure.

In synchronous systems the operators may begin action only at certain time instants determined by a central generator of

clock signals. The design of the control structure involves choosing the appropriate number and duration of clock intervals, and realizing a switching circuit that routes the clock signals to operators as required to implement the system's function.

In an asynchronous control structure,each operator in the data flow structure sends an acknowledge signal to the control structure to indicate that action by the operator has been completed. The acknowledge signals from operators are used directly in the control structure to initiate action by operators that become eligible for execution. In this way, initiation of an operator is delayed only until completion of those actions upon which correct functioning of the operator depends. No special generator of timing signals is used, the timing of system operation being determined by the durations of actions by the operators.

If the control structure of an asynchronous system will function correctly regardless of delays in its components and their interconnecting wires, the control structure is called a speed-independent circuit.

A system described by a logic diagram for a synchronous realization of it is both overspecified and underspecified. The particular choice of clock instants is irrelevant to the function performed by the system, but is essential for the diagram to have any meaning. Yet understandings between the specifier and implementer about timing of actions are necessary for unambiguous interpretation of the description. These understandings are not usually represented in a logic diagram. That a synchronous system is overspecified makes understanding or altering its function difficult; that it is underspecified makes design verification impossible in the absence of oversimplifying assumptions. The description of a system as a speed-indepedent circuit does not suffer these problems. Two parts of a speed-independent circuit are interconnected if, and only if, some action by one part is dependent on completion of some action by the other.

This reasoning shows that speed-independent implementation of digital systems is of particular interest when one desires assurance that a paper design will yield a correctly functioning system when translated into hardware. Speed-independent implementation is also attractive where a system is built from several interacting parts (there are no clocks in the subsystems to be synchronized), or where a system has much concurrent activity (which could only be slowed up by synchronizing action to common clock signals). Computer systems developed in the future are likely to have all of these characteristics.

The group has been studying schemes for representing systems so that conversion of the description into a speed-independent realization may be accomplished by a mechanical procedure with a guarantee that the resulting hardware will function correctly according to the description. In this way, the onerous task of debugging the hardware (as opposed to debugging the system description) would be largely eliminated. In

particular the faults that appear in hardware systems because of misunderstandings about the timing of signals would be avoided.

We are considering two classes of speed-independent circuits based on two assumptions regarding the origin of delays which must not affect correctness of system operation. Both classes of circuits are interconnections of primitive modules which may be individual gates or specific circuits realized in turn by the interconnection of simpler modules or gates.

In a type 1 circuit we assume that all interconnecting wires are sources of arbitrary delays. Thus a signal sent out by one module to two others may reach one module arbitrarily earlier than the other. In a type 2 circuit we assume that the output of a module may be delayed arbitrarily, but when an output of a module changes, the change is observed immediately by all modules to which the output is connected. The type 2 assumption is less restrictive, and is appropriate for circuits in which delays on interconnecting leads are negligible compared to delays within gates. This is normally the case within a semiconductor chip, for example. The more general type 1 assumption is appropriate for interconnections between standard parts where the designer does not know the mechanical arrangement of the parts.

A principal goal of our work is to find a finite set of practical modules with which it is possible to implement any digital system as a type 1 speed-independent circuit. In last year's report we described a collection of control modules adequate to implement any marked graph as a type 1 circuit. The complete set of control modules are also adequate for implementing free choice and simple Petri nets in the form of type 1 speed-independent circuits, and are convenient for defining control structures for complex digital systems.

The C-element of Muller [7] is a very important gate type for the construction of control modules. We have shown that the C-element cannot be implemented as a type 1 interconnection of AND, OR and NOT gates. In fact, there is very little that can be done by a type 1 speed-independent circuit using only AND, OR and NOT gates. These results are included in a paper by Dennis and Patil [8]. Since several basic control modules have type 1 realizations using NOT gates and C-elements, these results emphasize the importance of the C-element as a fundamental gate type for speed-independent circuits. More recently, Fred Furtek has defined a complete set of basic modules for the realization of general Petri nets as type 1 speed-independent circuits.

Our success in applying speed-independent design to control structures for digital systems has led us to investigate the applicability of the concept to complete digital systems. As an experiment, Dennis and Plummer developed a design for a fast counter that could be sampled repeatedly without interfering with continuation of counting. The design is a type 1

interconnection of as many identical stages as desired, each stage being a type 2 circuit using OR-gates, NOT-gates and C-elements. Commands to 'count' or to 'sample' flow through the stages of the counter from the least significant end changing or reading the bit held by each stage. In this way the speed of the counter is independent of the number of stages. The details of the design have been reported [8]. Bill Plummer designed and constructed an arbiter module to resolve conflicts between 'count' and 'sample' commands, and has prepared a paper on his work [9].

D. Base Language

The Group is working toward the definition of a common base language that could serve as a target representation for procedures translated from a variety of practical source languages, for example, FORTRAN, ALGOL and LISP. By specifying a formal interpreter for the base language and giving a precise description of the translation of source programs into base language programs, we would have a complete scheme for the formal definition of the semantics of programming languages in terms of a common set of semantic notions (those of the base language).

The motivation for this work is the design of computer systems in which the creation of correct programs is as convenient and easy as possible. A major factor in the convenient synthesis of programs is the ability to build large programs by combining simpler procedures or program modules, written independently, and perhaps by different individuals using different source languages. This ability of a computer system to support modular programming is called programming generality [10,11]. Programming generality requires the communication of data among independently specified procedures,and thus that the semantics of the languages in which these procedures are expressed must be defined in terms of a common collection of data types and a common concept of data structure.

We have observed that the achievement of programming generality is very difficult in conventional computer systems, primarily because of the variety of data reference and access methods that must be used for the implementation of large programs with acceptable efficiency. For example, data structures that vary in size and form during a computation are given different representations from those that are static; data that reside in different storage media are accessed by different means of reference; clashes of identifiers appearing in different blocks or procedures are prevented by design in some source languages, but similar consideration has not been given to the naming and referencing of cataloged files and procedures in the operating environment of programs. These limitations, on the degree of generality possible in computer systems of conventional architecture have led us to study new concepts of computer system organization through which these limitations on programming generality might be overcome.

In this effort, we are working at the same time on developing

the base language and on developing concepts of computer architecture suited to the execution of computations specified by base language programs. Thus our work on the base language is strongly influenced by hardware concepts derived from the requirements of programming generality [10].

We have chosen trees with shared substructures as our universal representation for information structures because we have found attractive hardware realizations of memory systems for tree-structured data. Jeffery Gertz [12] has considered how such a memory system might be designed as a hierarchy of associative memories. Also, the base language is intended to represent the concurrency of parts of computations in a way that permits their execution in parallel. One reason for emphasizing concurrency is that it is essential to the description of certain computations; for example, when a response is required to whichever one of several independent events is first to occur. Furthermore, we believe that exploiting the potential concurrency in programs will be important in realizing efficient computer systems that offer programming generality. This is because concurrent execution of program parts increases the utilization of processing hardware by providing many activities that can be carried forward while other activities are blocked, pending retrieval of information from slower parts of the computer system memory.

When the meaning of algorithms, expressed in some programming language,has been specified in precise terms, we say that a formal semantics for the language has been given. A formal semantics for a programming language generally takes the form of two sets of rules; one set being a translator, and the second set being an interpreter. The translator specifies a transformation of any well-formed program expressed in the source languge (the concrete language) into an equivalent program expressed in a second language -- the abstract language of the definition. The interpreter expresses the meaning of programs in the abstract language by giving explicit directions for carrying out the computation specified by any well-formed abstract program.

It would be possible to specify the formal semantics of a programming language by giving an interpreter for the concrete programs of the source language; the translator is then the identity transformation. Yet the inclusion of a translator in the definition scheme has important advantages. For one, the phrase structure of a programming language,viewed as a set of strings on some alphabet,usually does not correspond well with the semantic structure of programs. Thus, it is desirable to give the semantic rules of interpretation for a representation of the program that more naturally represents its semantic structure. Furthermore, many constructs present in source languages are provided for convenience rather than as fundamental linguistic features. By arranging the translator to replace occurrences of these constructs with more basic constructs, a simpler abstract language is possible, and its interpreter can be made more readily understandable and, therefore, more useful as a tool for the design and specification of

computer languages and systems.

Our thoughts on the definition of programming languages in terms of a base language are closely related to the formal methods developed at the IBM Vienna Laboratory [13] and which derive from the ideas of McCarthy [14] and Landin [15].

For the formal semantics of programming languages, a general model is required for the data on which programs act. We regard data as consisting of elementary objects, and compound objects formed by combining elementary objects into data structures. Elementary objects are data items whose structure in terms of simpler objects is not relevant to the description of algorithms. For the purposes of this discussion, the class E of elementary objects is

$$E = Z \cup R \cup W$$

where

Z = the class of integers
R = a set of representations for real numbers
W = the set of all strings on some alphabet

Data structures are often represented by directed graphs in which elementary objects are associated with nodes, and each arc is labelled by a member of a set S of selectors. We will use integers and strings as selectors:

$$S = Z \cup W$$

In the class of objects used by the Vienna group, the graphs are restricted to be trees, and elementary objects are associated only with leaf nodes. We have used a less restricted class so an object may have distinct component objects that share some third object as a common component.

> Let E be a class of elementary objects, and let S be a class of selectors. An object is a directed acyclic graph having a single root node from which all other nodes may be reached over directed paths. Each arc is labelled with one selector in S, and an elementary object in E may be associated with each leaf node.

An example of an object is shown in Fig. 8. Leaf nodes having associated elementary objects are represented by circles with the element of E written inside: Integers are represented by numerals, strings are enclosed in single quotes, and reals have decimal points. Other nodes are represented by solid dots, with a horizontal bar if there is more than one emanating arc.

The node of an object reached by traversing an arc emanating from its root node is itself the root node of an object called a component of the original object. The component object consists of all nodes and arcs that can be reached by directed paths from its root node.

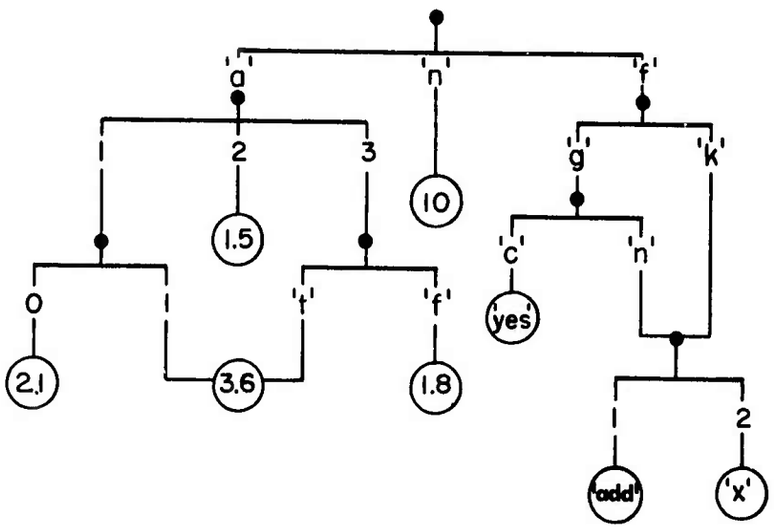


FIG. 8.

Some of us prefer to generalize this class of objects in two ways:

1) by permitting data values to be associated with any node of the graph of a structure

and

2) by permitting the graph to contain directed cycles.

Whether to permit cycles in the structured data objects of the base language is an important unresolved issue. Some considerations bearing on this matter are discussed in a later paragraph of this report.

Figure 9 shows how source languages would be defined in terms of a common base language. Concrete programs in source languages (L1 and L2 in the Figure) are defined by translators into abstract programs of the base language. For this to be effectively possible, the structure of abstract programs cannot reflect the peculiarities of any particular source language, but must provide a set of fundamental linguistic constructs in terms of which the features of these source languages may be realized. The translators themselves should be specified in terms of the base language, probably by means of a specialized source language. Formally, abstract programs in the base language, and states of interpreter are elements of the class of objects defined above.

The structure of states of the interpreter for the base language is shown in Fig. 10. Since we regard the interpreter for the base language as a complete specification for the functional operation of a computer system, a state of the interpreter represents the totality of programs, data, and control information present in the computer system. The universe is an object that represents all information present in the computer system when the system is idle, that is, when no computation is in progress. The universe has data structures and procedure structures as constituent objects. Any object is a legitimate data structure; for example, a data structure may have components that are procedure structures. A procedure structure is an object that represents a procedure expressed in the base language. It has components which are instructions of the base language, data structures, or other procedure structures. So that multiple activations of procedures may be accommodated, a procedure structure remains unaltered during its interpretation.

The local structure of an interpreter state contains a local structure for each current activation of each base language procedure. Each local structure has as components, the local structures of all procedure activations initiated within it. Thus the hierarchy of local structures represents the dynamic relationship of procedure activations.

The control component of an interpreter state is an unordered set of sites of activity. A typical site of activity is

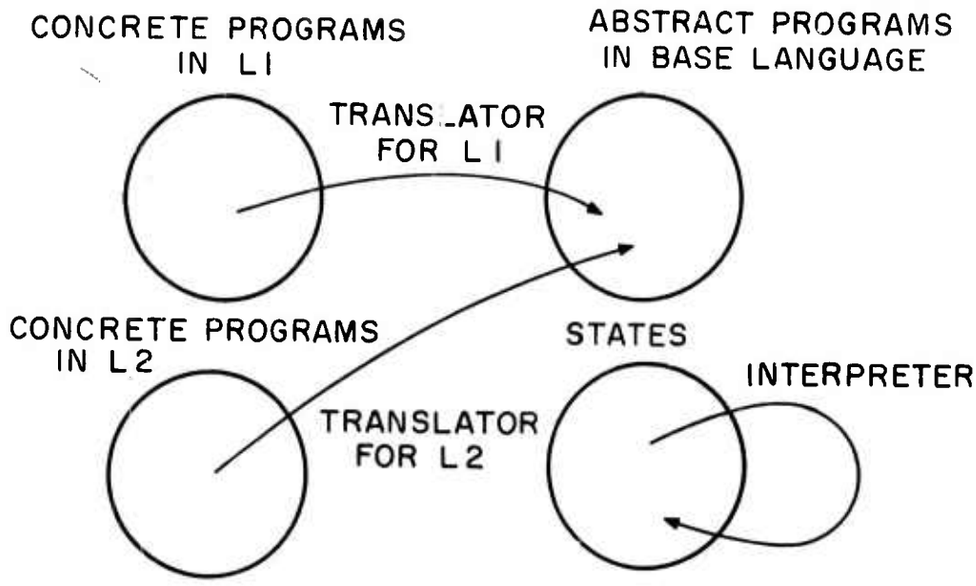

CONCRETE PROGRAMS
IN L1
ABSTRACT PROGRAMS
IN BASE LANGUAGE
TRANSLATOR
FOR L1
CONCRETE PROGRAMS
IN L2
STATES
INTERPRETER
TRANSLATOR
FOR L2

FIG. 9.

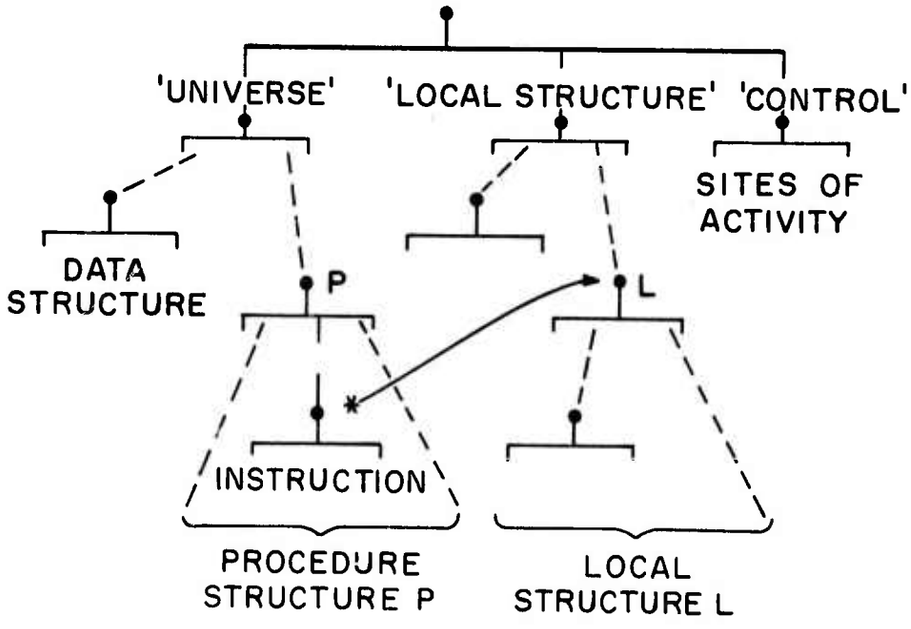

'UNIVERSE'
'LOCAL STRUCTURE'
'CONTROL'
SITES OF
ACTIVITY
DATA
STRUCTURE
P
L
*
INSTRUCTION
PROCEDURE
STRUCTURE P
LOCAL
STRUCTURE L

FIG. 10.

represented in the figure by an asterisk at an instruction of procedure P and an arrow to the local structure L for some activation of P. Since several activations of a procedure may exist concurrently, there may be two or more sites of activity involving the same instruction of some procedure, but designating different local structures. Also, within one activation of a procedure, several instructions may be active concurrently; thus asterisks on different instructions of a procedure may have arrows to the same local structure.

Each state transition of the interpreter executes one instruction for some procedure activation, at a site of activity selected arbitrarily from the control of the current state. Thus the interpreter is a nondeterministic transition system. In the state resulting from a transition, the chosen site of activity is replaced by zero or more new sites of activity according to the sequencing rules of the base language.

Interpretation of a procedure involves two objects, the procedure structure P and an argument structure A. The argument structure is formed by the calling procedure activation and contains, as component objects, all information (other than P) required by the activation of P. In particular, the actual parameters of the procedure activation are components of A. In this view of procedure execution, no meaning is given to nonlocal references occurring within a procedure structure. Thus no side effects of procedure executions are possible. Unless procedure P modifies part of its own procedure structure, it defines an algebraic operation on the class of all objects.

A subject of major importance to us is the representation of concurrent activities in the base language. Consideration of concurrency brings in the issue of nondeterminacy -- the possibility that computed results will depend on the relative timing with which the concurrent activities are carried forward. The ability of a computer user to direct the system to carry out computations with a guarantee of determinacy is very important. Most programs are intended to implement a functional dependence of results on inputs, and determinism is essential to the verification of their correctness.

There are two ways of providing a guarantee of determinacy to the user of a computer system. They are distinguished according to whether or not the class of base language programs is constrained through design of the interpreter to describe only determinate computations. If this is the case, then any abstract program resulting from compilation will be deterministic in execution. Furthermore, if the compiler is itself a determinate procedure, then each translatable source program represents a determinate procedure. On the other hand, if the design of the interpreter does not guarantee determinacy of abstract programs, determinacy of source programs, when desired, must be ensured by the translator.

E. Program Graphs

We are considering two approaches to represent the relationships

among instructions of a procedure structure:

1. A conventional form in which the instructions of each procedure structure are selected by successive integers, and instructions are executed sequentially except when a conditional transfer of control directs execution to a new instruction sequence.

In this form, concurrency is represented by fork instructions where activity splits into two concurrent streams and join instructions where two streams of activity merge into one.

2. A data flow form in which execution of an instruction is controlled by the availability of the data values required for its execution. For example, execution of an add instruction would be enabled as soon as the values of both operands have been computed.

Concurrency is inherent in a data flow representation since the creation of a computed value may enable several instructions. The data flow representations we are investigating are variations and extensions of the program graphs introduced by Rodriguez [16]. We shall illustrate our present thoughts regarding data flow representations by presenting program graphs for several programs. Consider the program

```
begin
     v := t - x; w := x - u
     if v > w then y := w - 2 else y := v + 3
     if y > 0 then z := y ↑ 2 else z := 0
end
```

A conventional machine level representation would be:

```
begin
     fork ℓ1              ℓ3:  w - 2 → y
     t - x → v            ℓ4:  if y > 0 goto ℓ5
     goto ℓ2                   0 → z
ℓ1:  x - u → w                 goto ℓ6
ℓ2:  join                 ℓ5:  y ↑ 2 → z
     if v > w goto ℓ3      ℓ6:  end
     v + 3 → y
     goto ℓ4
```

A program graph for this program is shown in Fig. 11. The nodes of the program graph include functional operators drawn as circles, predicate operators drawn as diamonds and two special node types, gate and merge, that perform control functions. The links may be thought of as conveying tokens

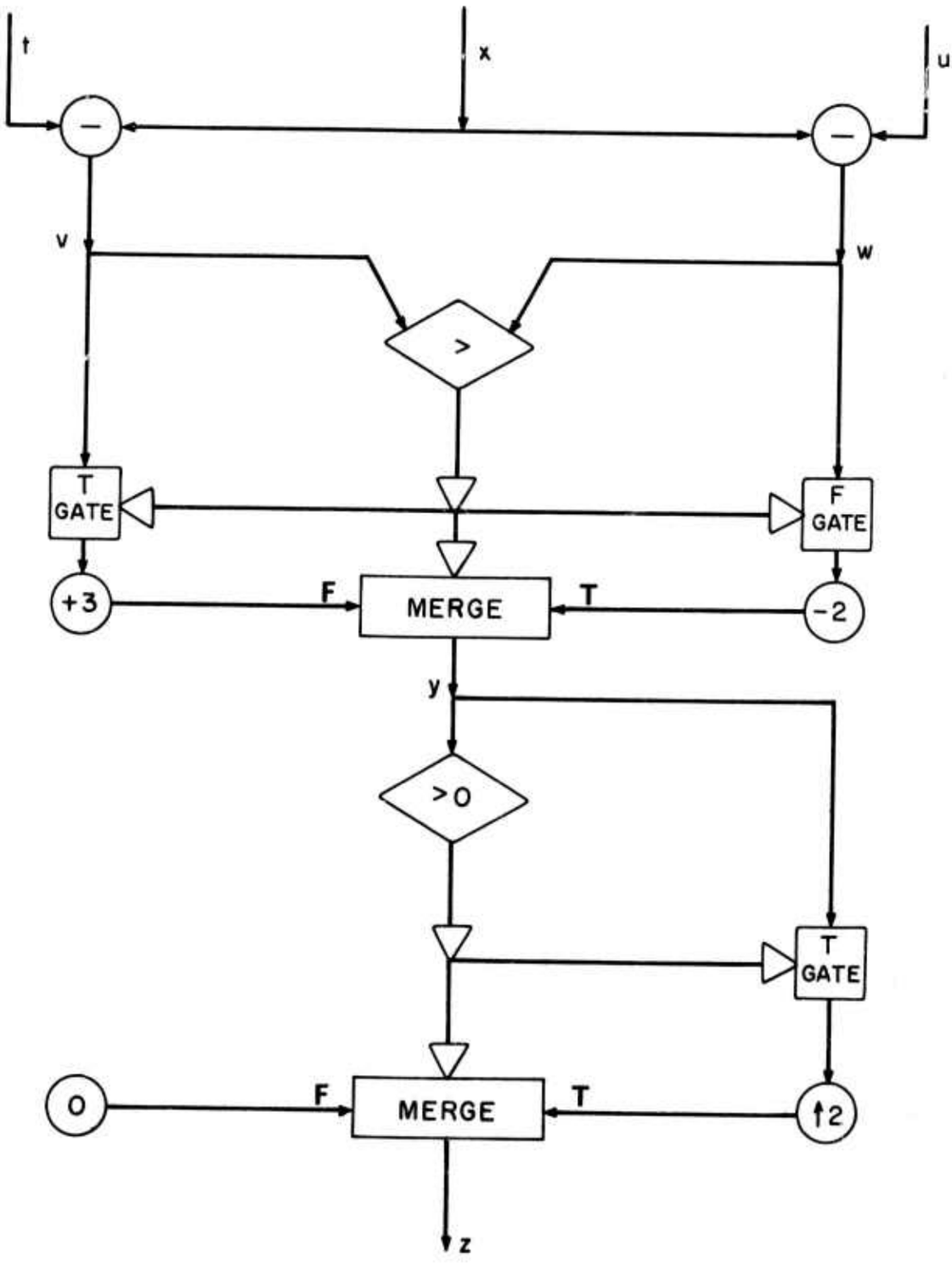


FIG. 11.

between nodes of the diagram as in a Petri net. Here the tokens have information associated with them. Tokens arriving at or leaving functional operators, and those arriving at predicate operators convey values (numbers for example); these links are drawn with small solid arrows. Tokens leaving a predicate operator convey decisions (true or false) to gate nodes of the diagram; these links are drawn with open arrowheads. We assume the net operates in a safe manner, that is, tokens do not overtake one another, nor do they accumulate at nodes. This may be ensured by acknowledge signals transmitted in the reverse direction over each link. Thus a value link may be represented in a Petri net by a pair of places: a place (drawn as a square box) through which tokens with attached values move from source node to destination, and an ordinary place through which "empty" tokens are returned to the source node. Decision links may be conveniently represented by three places through which ordinary tokens (not bearing values) move. A token arriving at the place labeled t signals a true decision; a token arriving at the place labeled f signals a false decision.

When a link goes to two or more destinations, tokens are replicated at each branch point so that tokens with identical information are sent to each node. The branch points act like wye modules, and await acknowledgment signals from each destination before returning an empty token to the source node.

The gate and merge control nodes are needed so that decisions made by predicate operators may affect the pattern of data flow through functional operators of the program graph. A T-gate node permits a value-bearing token to pass through for each true decision received on the decision link. Whenever a false decision arrives the value-bearing token is not forwarded. In either case the gate node acknowledges both tokens received, and when a gate forwards a token, it waits for acknowledgment before forwarding another value-bearing token. The behavior of a gate node is described in Fig. 12. The arrival of a true decision leads to forwarding of a value token from link 1 to link 2. Arrival of a false decision causes a value arriving on link 1 to be acknowledged and discarded. An F-gate node is identical to the T-gate except that the sense of the decision is reversed.

A merge node permits values sent over its output link to originate from different sources according to decisions made during computation. The value sent over the output link is forwarded from the T- or F-labeled input value link according as the decision received is true or false. A Petri net for the switch node is shown in Fig. 13.

Next we give an example showing how iterative programs can be represented as program graphs:

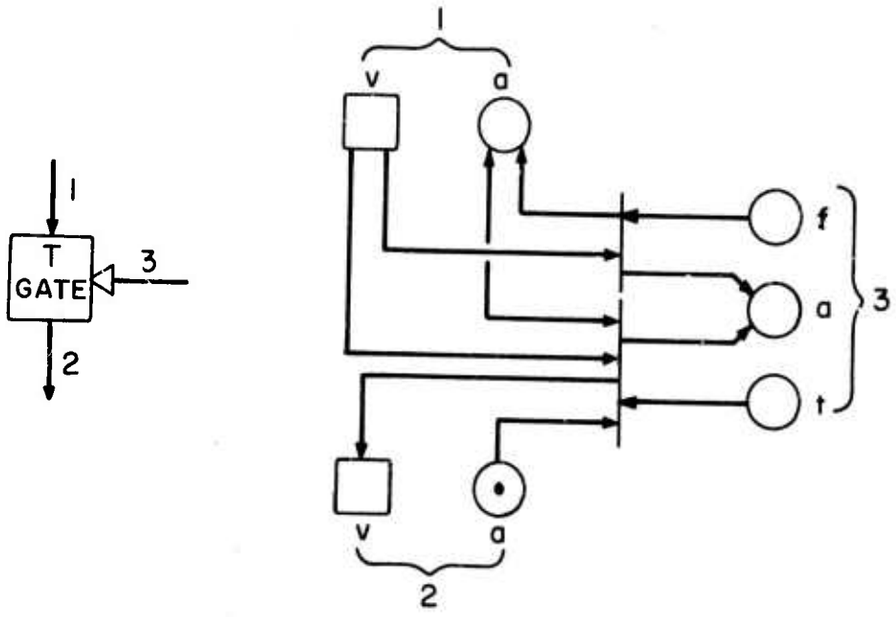


FIG. 12.

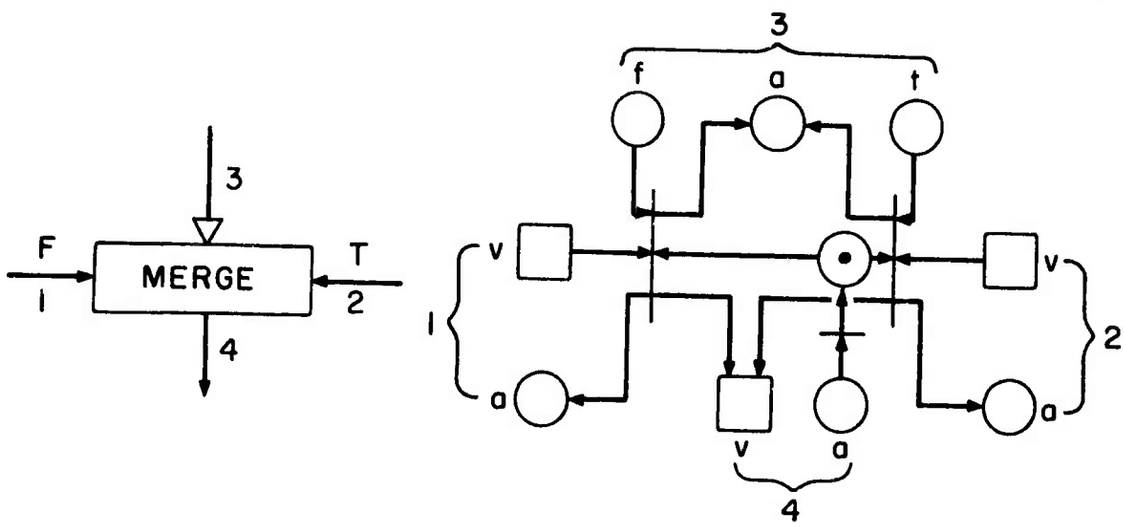


FIG. 13.

```
begin
     y := x
     v := 0
     while p(w,v) do
        begin
            v := f(v); y := g(y)
        end
     z := y
end
```

Noting that the two statements of the body of the iteration may be performed concurrently, a conventional representation would be similar to this:

```
begin
     x → y
     0 → v
ℓ1:  if p(w,v)goto ℓ4
     fork ℓ2
     f(v) → v
     goto ℓ3
ℓ2:  g(y) → y
ℓ3:  join
     goto ℓ1
ℓ4:  y → z
     end
```

A data flow version of the program is provided in Fig. 14. Two of the merge nodes serve as the junctions through which initial values and intermediate values flow to the functional operators of the body of the while loop. The predicate operator requires one copy of the value of variable w for each test of the predicate p. These copies are generated by the center merge node, and the associated gate node. Initiation of operation of the program graph requires arrival of a false decision at the decision input link of each of the three merge nodes. This is provided by the F-buff node which is a buffer for decisions that sends a false decision as its initial output, (Fig. 15.).

An important result of Suhas Patil [17] concerning interconnections of determinate systems can be applied to program graphs formed from the node types used in these two examples. We conclude that any such program graph is a determinate representation of a program. This class of program graphs is a revision of the class studied earlier by Rodriguez, and is

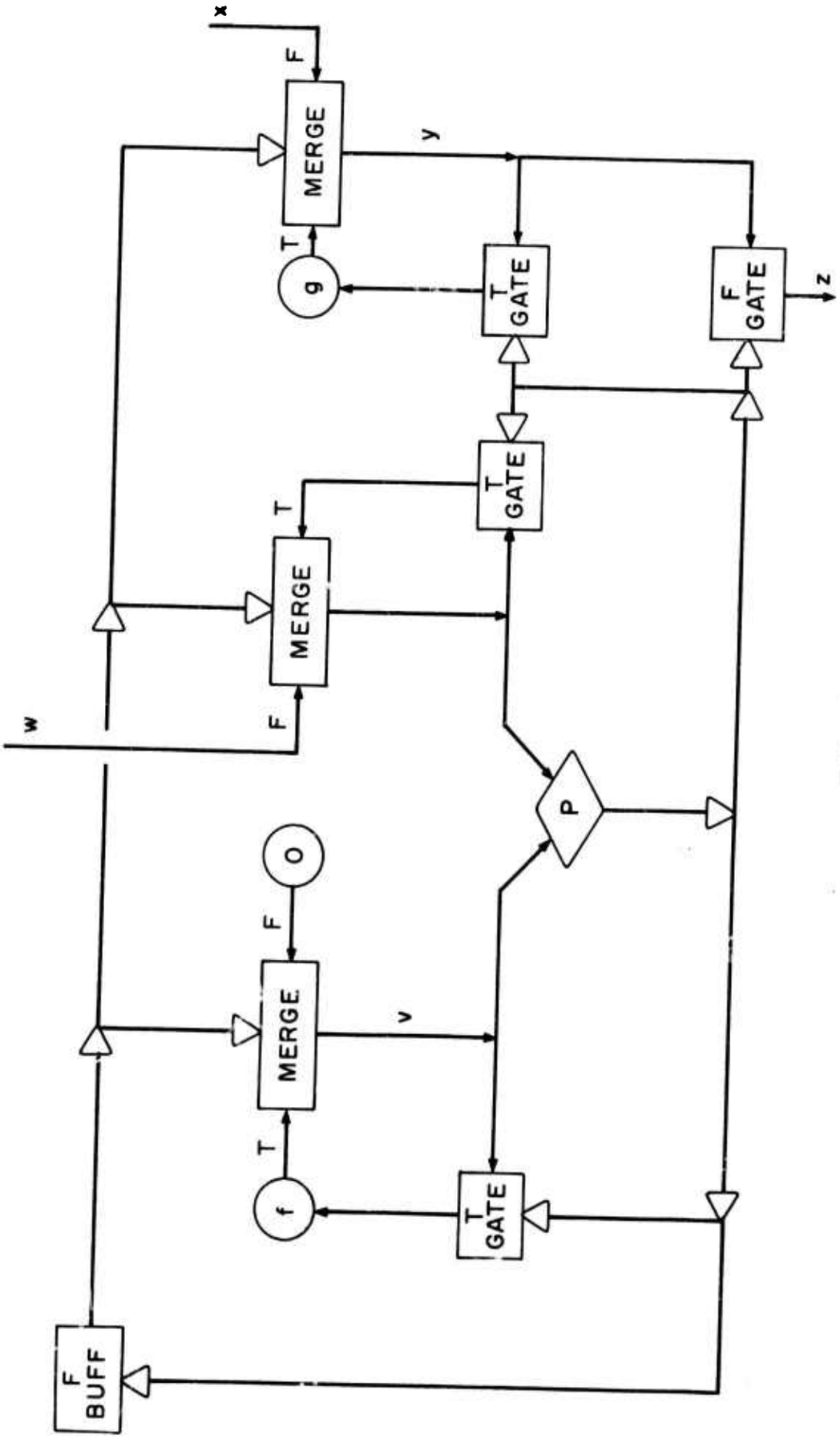

x
F
MERGE
y
T
g
T
GATE
F
GATE
z
T
GATE
T
MERGE
w
F
P
O
F
MERGE
v
T
f
T
GATE
F
BUFF

FIG. 14.

simpler as a result of our improved understanding of concurrent activities. We expect that future developments in the theoretical study of Petri nets will contribute significantly to the building of a satisfactory theory of program graphs.

Jack Dennis has formulated a class of program graphs suitable for representing certain computations on structured data [10]. These program graphs were limited in that no provisions were made for conditional execution of subgraphs or for iterative computation. We expect to combine the concepts developed in this class with those of Rodriguez to obtain a general class of program graphs encompassing,say,all ALGOL 60 programs. Our final example illustrates the form this class of program graphs may take.

```
procedure (a,b,n)
    begin
    y := 0
    for i := 1 step 1 through n do
        y := y + a[i] x b[i]
    return y
    end
```

The input data for this procedure will be represented by the argument structure shown in Fig. 16, having components for the three formal parameters of the procedure. In the program graph shown in Fig. 18, a third kind of link is used and is drawn as a heavy line with a solid arrowhead. Tokens passing on these links convey access to objects. Execution is initiated by arrival of a token at the root node P of the program graph. This token carries access to an argument structure of the form shown. Four new node types are used, (Fig. 17). The select x node converts access to an object into access to the x-component of the object. These nodes are used to obtain access to the components of the argument structure. The second form of select node uses the integer received on link 3 to select the component object. The value node converts access to an elementary object into the value of the object. Finally, the assign node receives a data value on link 2 and transforms the object conveyed on link 1 into an elementary object having that value.

The repeat nodes in this program graph generate multiple copies of tokens conveying access to the same object, in this case the actual parameters of the scalar product procedure. One token is sent over the output link for each true decision received on the decision link. Acknowledgment is not given on the input data link until a false decision is received, whereupon the node resets and waits for the arrival of new data.

This program graph is determinate, yet we cannot guarantee the determinacy of any program graph constructed from all node types introduced here. We would like to find a set of program

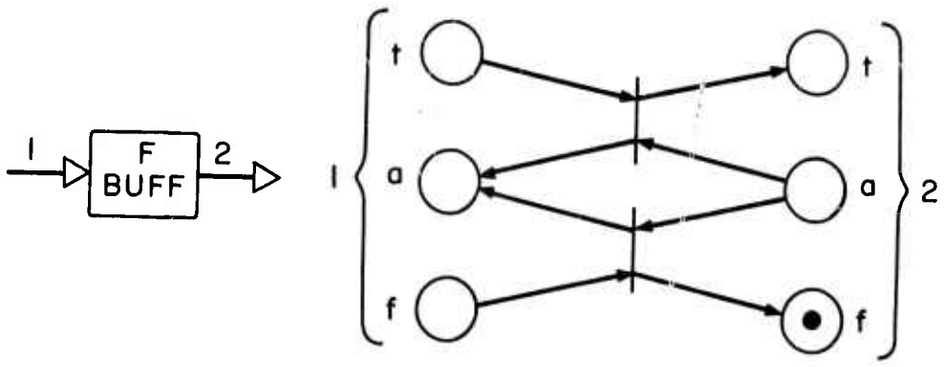


FIG. 15.

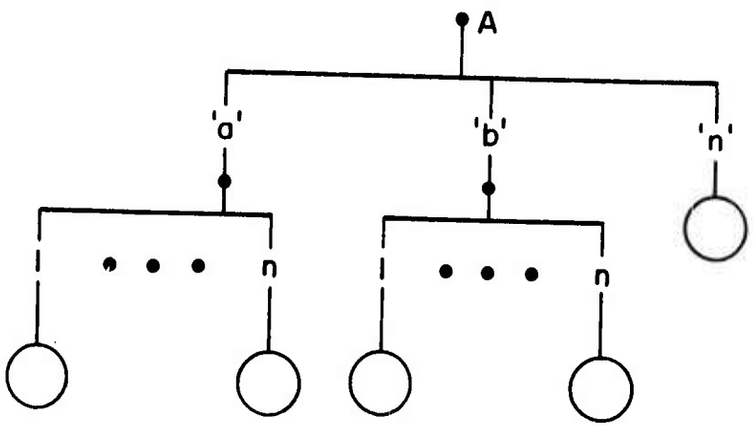


FIG. 16.

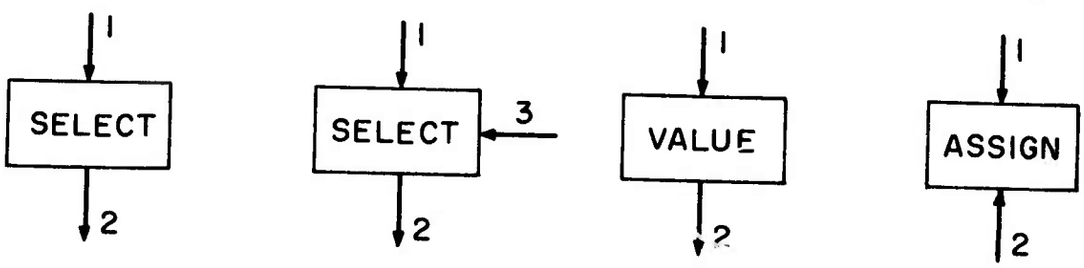


FIG. 17.

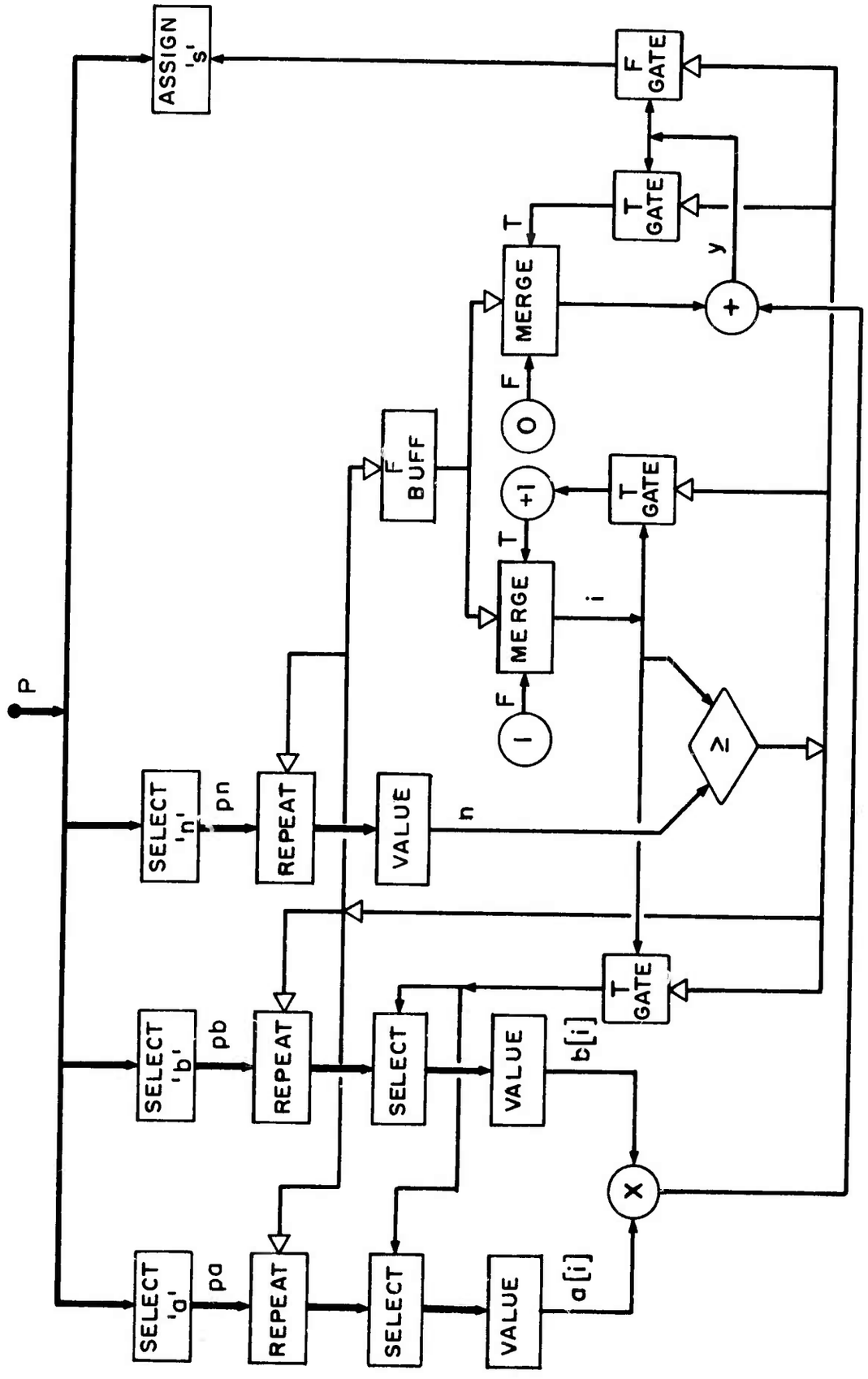

ASSIGN 's'
F GATE
T GATE
y
T
MERGE
+
F
0
F BUFF
+1
T GATE
T
MERGE
i
F
1
≥
P
SELECT 'n'
pn
REPEAT
VALUE
n
T GATE
SELECT 'b'
pb
REPEAT
SELECT
VALUE
b[i]
X
SELECT 'a'
pa
REPEAT
SELECT
VALUE
a[i]

FIG. 18.

graph node types and a condition on their interconnection, such that the program graphs satisfying the condition are determinate and include representations for a wide variety of programs.

Certain computations are more naturally expressed in data flow terms than in conventional form. A typical example is a situation in which several independent activities generate and consume units of data exchanged among themselves. Suppose a computation is performed by two interconnected modules, (Fig. 19.). Module 1 takes an initial value x from data cell a and generates a sequence of values $y_0, y_1, \ldots y_n$ that are forwarded to module 2 through data cell b. Module 2 processes these values as they become available, and, when all values have been processed, puts a cumulative result z in cell c. Let the computations performed by modules 1 and 2 be described by the following relations where f and g denote unspecified functions.

$$y_0 = f(x) \qquad w_0 = 0$$

$$y_1 = f(y_0) \qquad w_1 = g(y_0, w_0)$$

$$\vdots \qquad\qquad \vdots$$

$$y_k = f(y_{k-1}) \qquad w_k = g(y_{k-1}, w_{k-1})$$

$$z = g(y_k, w_k)$$

A program graph for this computation is shown in Fig. 20. The predicate p is applied to each value y_i by both modules to determine when the last value of a sequence has been processed:

$$p(y_i) = \underline{\text{true}}, \quad i = 1, \ldots, k - 1$$

$$p(y_k) = \underline{\text{false}}$$

Note that this program graph allows the two modules to act concurrently and is formed simply by connecting together program graphs that represent the two modules. Furthermore, the incorporation of a first in-first out queue in the connecting link would permit module 1 to continue generating values even when module 2 has not had enough time to use up the previous values. The addition of queues does not require any change in the representations of the modules. These properties are not shared by other representations such as co-routines or processes inter-communicating by means of semaphores. Further discussion of these points appears in a recent paper by Jack Dennis [18].

Program graphs are an attractive representation for procedures expressed in the base language because the possibilities for concurrent execution of instructions are exhibited in a natural way. Program graphs represent many procedures in their maximum parallel form. Also, it is easy to impose constraints on

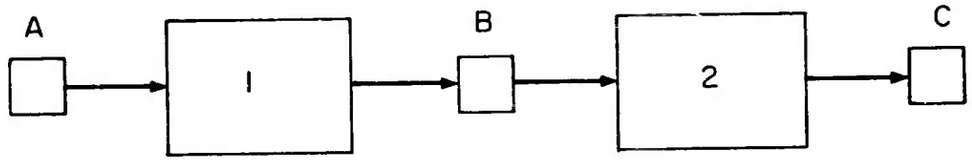


FIG. 19

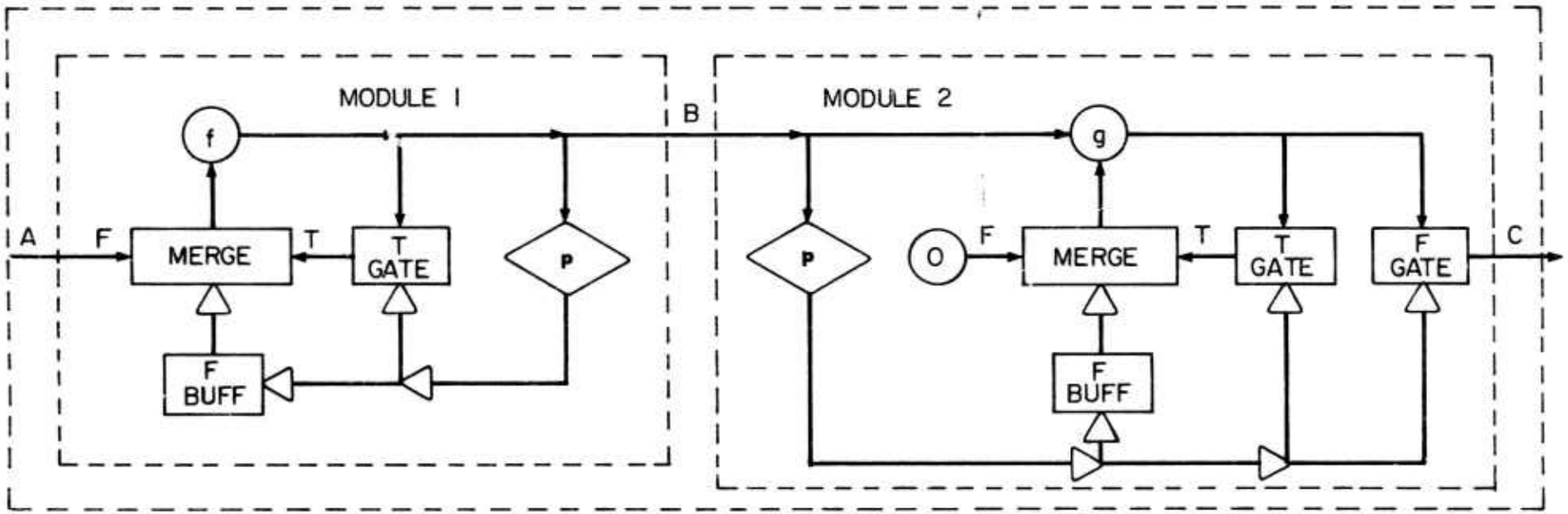


FIG. 20

program graphs such that determinate execution is assured without restricting the class of determinate procedures that can be expressed. Finally, we have found that considering program graphs as a machine level representation leads to interesting concepts for the structure of highly parallel computers [10].

F. Translation of Block-Structured Languages

Many important programming languages for practical computation are block structured; the texts of blocks and procedures are nested, and identifiers appearing in one text may refer to variables declared in other texts. We do not plan to include in the base language provision for directly representing reference by a procedure to external objects. Therefore, we must show how the execution of block-structured programs may be effected through translation into the base language and execution by the base language interpreter. The following discussion outlines one way in which this may be accomplished -- a way that seems attractive in view of the concepts of computer organization we are investigating.

Consider the program shown in Fig. 21. This program has the block structure shown; the main block P encloses a procedure declaration P and a block Q. Upper case letters are used to identify the texts of blocks or procedures.

If T is a text (block or procedure declaration) of a program, let B(T) be the set of identifiers occurring in T that are locally declared. Let X(T) be the set of identifiers occurring in T, or any text nested within T, that refer to variables declared outside T. For the above program we have

$$B(P) = \{y, z, f\} \quad B(F) = \{x\} \quad B(Q) = \{y\}$$
$$X(P) = \emptyset \quad X(F) = \{y\} \quad X(Q) = \{f\}$$

Since non-local references are excluded in the base language, we need a scheme for making variables accessed by non-local reference in the block-structured program accessible through the argument structure in the base language representation. We will discuss one method of doing this, details of which are given in a recent paper by Jack Dennis [19]. To illustrate this scheme consider the computation of apply p (4). As objects, the procedure structure P and the local structure L(P) at the beginning of the computation will be as shown in Fig. 22. Texts F and Q are represented as components of the object representing text P. The local structure for the activation of P has one component for each identifier in the set $B(F) \cup X(F)$.

The first step is execution of the declaration of text F. This gives the procedure identifier f a value called a closure of the text F (Fig. 23). The C·T-component of the closure is the text of procedure F and is shared with the procedure structure P. The C·E-component of the closure links identifiers in X(F) to the value these identifiers have in the current procedure activation. Thus the identifier y shares the value 4 with y in L(P).

```
P   p := PROCEDURE(y)
    BEGIN REAL y,z
F   f :=  PROCEDURE(x)
          BEGIN REAL x
          y := y + x
          END
Q   q :   BEGIN REAL y
          y := 1
          APPLY f(y)
          END
    z :=  y ↑ 2
    RETURN z
    END
```

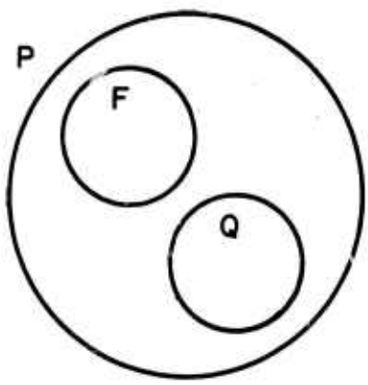
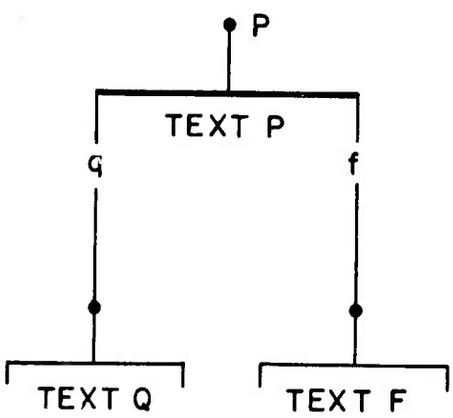


FIG. 21.



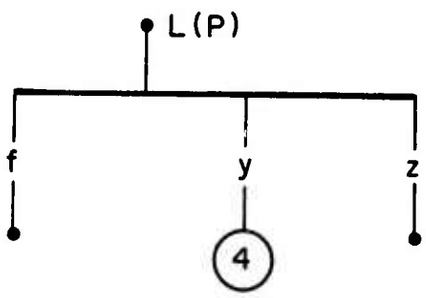


FIG. 22.

Entering block Q may be treated as though it were a procedure without parameters. A new local structure L(Q) is formed and made inferior to L(P), (Fig. 24.). This new local structure has a component for each identifier in $B(Q) \cup X(Q) = \{f, y\}$. Identifier f is external, so it is given the same meaning as f in L(P).

After y in L(Q) is assigned the value 1, the closure of F is applied to an argument structure having a 1-component of 1 and an E·y - component that conveys to F the correct meaning of its external identifier, (Fig. 25.).

The meanings of identifiers x and y in text F are established as in the case of Text Q. Since y is in X(F) it is linked to the E·y - component of the argument structure. Since x identifies the first formal parameter of text F, it is linked to the 1-component of the argument structure. In this way, execution of the assignment in text F correctly updates the value of y in the local structure L(P), (Fig. 26.).

G. Cycles in Structures

The class of objects defined earlier does not permit directed cycles to occur in the graph of an object. The desirability of this restriction on the class of objects has been the subject of considerable study and discussion. Arguments against permitting cycles include these:

1. Cyclic structures do not seem essential to the representation of the structured data types of current important source languages.

2. When cycles occur in linked list structures, they can usually be considered part of an implementation rather than an essential aspect of the data structures being represented.

3. The presence of cycles in objects makes it difficult to exploit the concurrency of parts of an algorithm.

The principal arguments in favor of permitting cycles are:

1. Generality of data structures should not be arbitrarily restricted.

2. Cyclic structures are important for representing certain kinds of data.

3. Implementation of a base language using cyclic structures will not present great difficulty.

We have studied two definitive questions to develop better understanding of the importance of cyclic data structures to the base language.

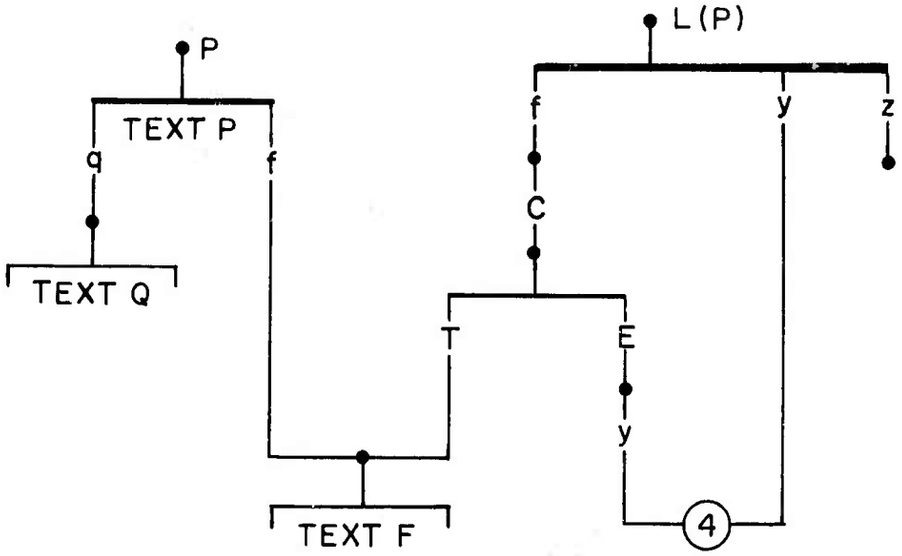


FIG. 23.

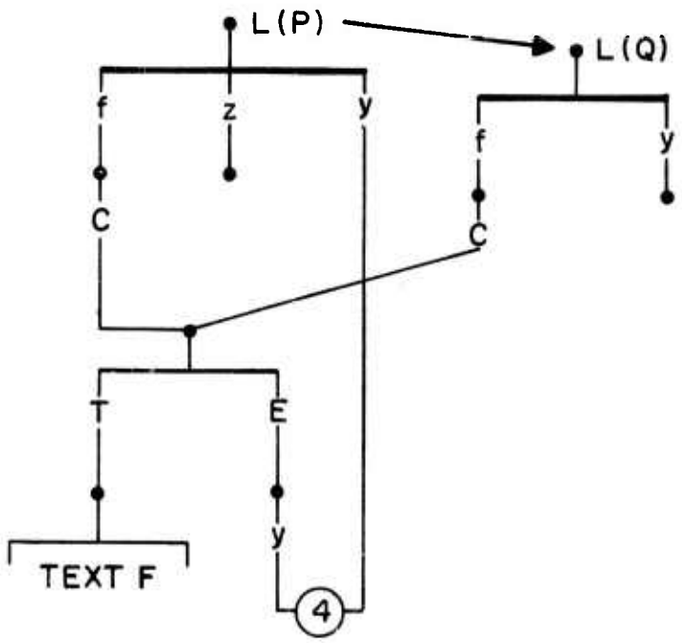


FIG. 24.

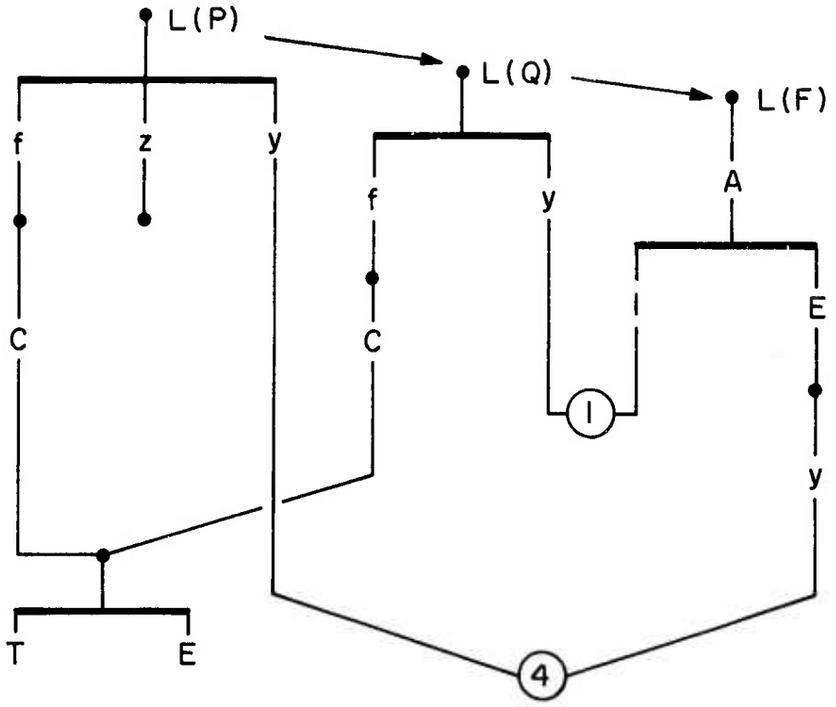


FIG. 25.

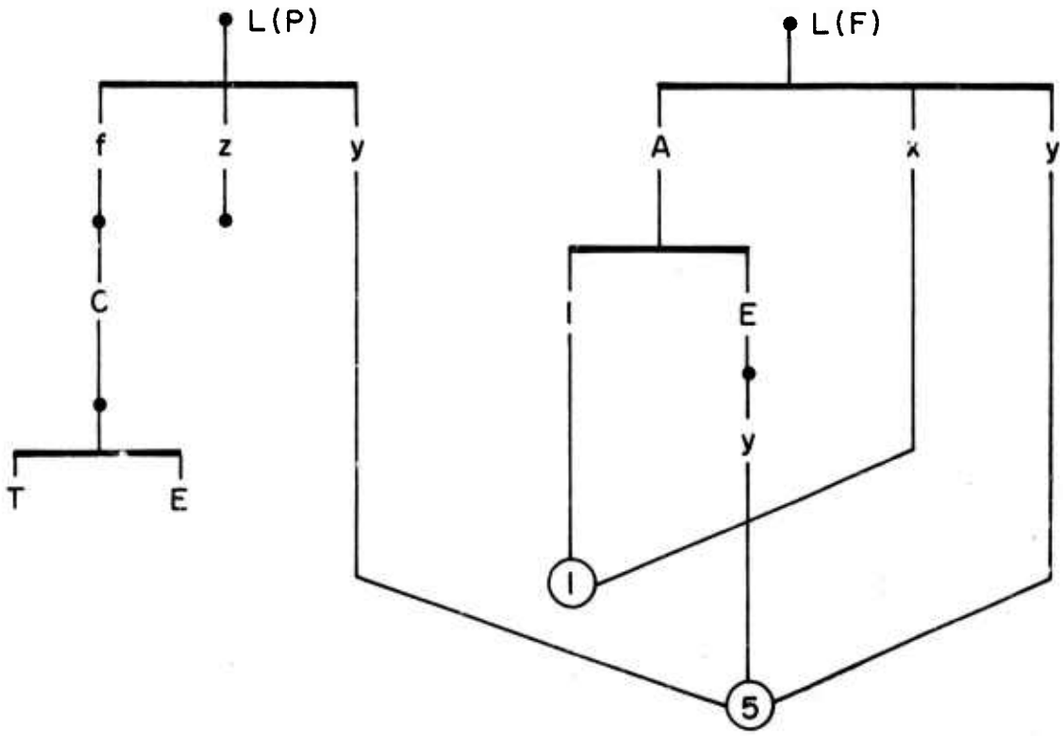


FIG. 26.

```
P  p := PROCEDURE(u)
   BEGIN
      F  f := PROCEDURE(x); INTEGER x
         BEGIN
            IF x = 0 THEN RETURN 1
            x := g(x)
            z := APPLY f(x)
            RETURN z
         END
      APPLY f(u)
   END
```

FIG. 27.

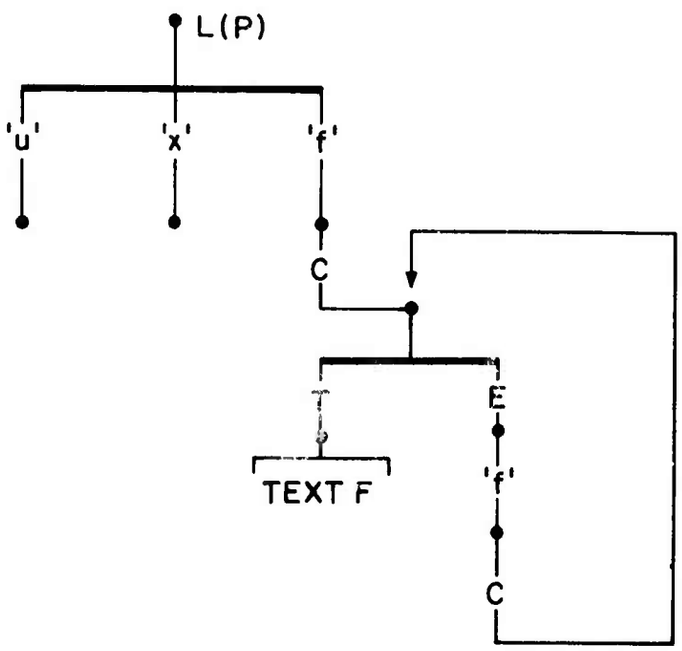


FIG. 28.

One study [19] concerns how cycles can arise during execution of block-structured programs according to the scheme outlined earlier. Consider the program shown in Fig. 27.

This program consists of a procedure declaration F which contains an application of itself. Interpretation of the declaration as described above assigns identifier f a value which is a closure of F, and in which f appears as an external reference. This creates a cycle in the local structure L(P), (Fig. 28).

We have found that many block-structured programs can be rewritten so they accomplish the original computation, but without the creation of cycles. The principle is to convey closures to and from a procedure activation by passing them as parameters or results rather than by external references. For example, the program given above becomes:

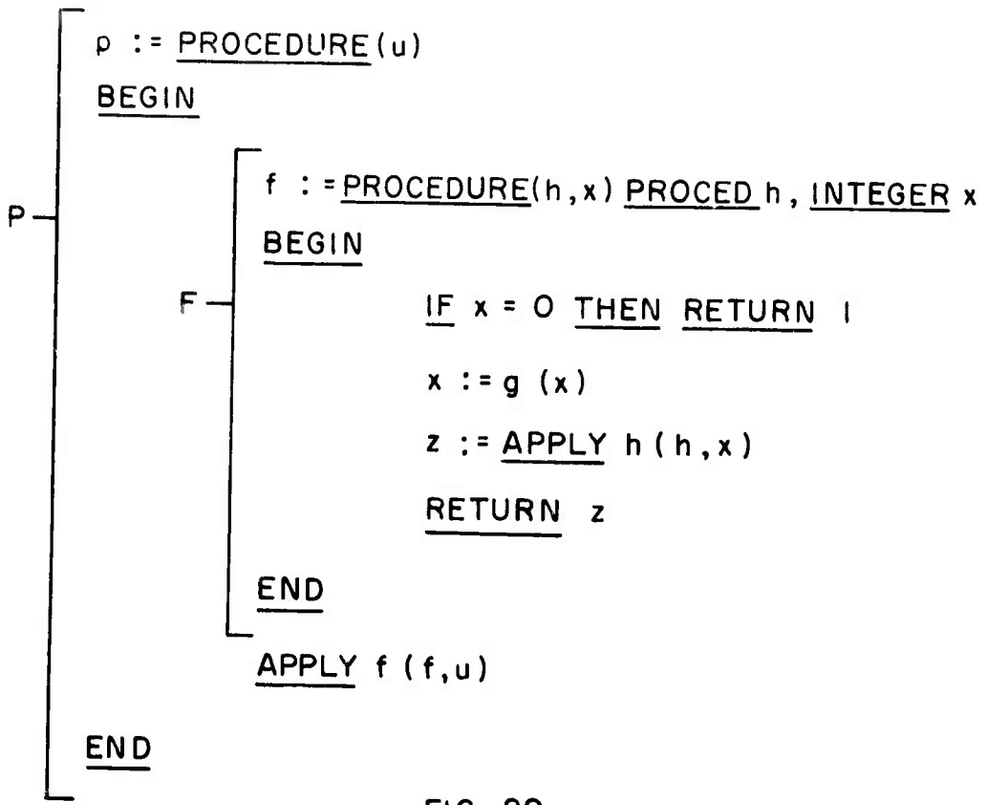


FIG. 29.

This raises some interesting questions. In particular, we would like to develop a general method for rewriting lock-structured programs so that cycles will not arise during execution.

The second study by Ian Campbell-Grant [20] investigated an execution model for multiprocess computations that operate on a data base represented as an arbitrary directed graph. The arcs of the graph represent structural relations among data items associated with the nodes. In this model each process may hold several pointers by which it may access the data base. Each pointer has an associated access control indicator having one of the three values:

R read access
WD write data access
WS write structure access

If a pointer carries R-access to a node, the process may apply the pointer to read (but not alter) the data associated with the node. The process may also obtain a pointer with R-access to any node that can be reached over a directed path in the data base from a node for which it holds R-access. A pointer carrying WD-access to a node permits the process to alter the data associated with the node, and to obtain a pointer with WD-access to any node accessible from the given node. A pointer carrying WS-access to a node permits a process to modify the graph of the data base by adding or deleting arcs within the subgraph formed by all arcs that can be traversed via directed paths starting from the given node. The three kinds of access are cumulative, that is, WD-access includes the privileges of R-access, and WS-access includes the privileges of R-access and WD-access.

The objective of this study was to show how constraints can be implemented in an execution model so that any computation carried on by a set of interacting processes would be determinate. For this purpose, a computation is regarded as determinate if it can never happen that two processes apply pointers to the same data base node concurrently, unless both processes possess only R-access.

The scheme used to ensure determinism involves a set of constraints. Each constraint is an ordered pair (A, B) where A and B are pointers held by distinct processes 1 and 2. The constraint (A, B) signifies that application of pointer B by process 2 must wait until process 1 reduces its access privilege for pointer A.

By executing certain instructions defined for the model, a process may: access nodes by following directed paths in the data base; create and terminate subsidiary processes; and apply pointers to read and write the data associated with accessible nodes of the data base. The execution rules for each instruction type includes specification of how the constraint set must be modified. Campbell-Grant has shown that the relation graph defined by the set of constraints will always be acyclic throughout any multiprocess computation by his model. In consequence, the following condition will always be satisfied, where the predicate struct (X,Y) is true, if and only if, there is a node in the data base reachable over directed paths from the nodes designated by pointers X and Y:

> If pointers A and B are held by distinct processes and struct (A,B) = true then access (A) = R and access (B) = R or one of (A, B) or (B, A) is in the constraint set.

This is sufficient to guarantee determinate computation.

H. Computers and People

When computers are used in any facet of the operation of society, the specific technical characteristics and capabilities of the computer system employed constrain and significantly influence the behavior of the larger system comprising hardware, software and people. We have learned by now that computer hardware should be designed and evaluated in the context of the software that provides the interface with the users. We must now learn how to design and evaluate computer systems in the context of the community of people that is affected by their use.

Two related problem areas require attention. One concerns the interaction between characteristics of computer systems and the individual and collective behavior of the people affected by their use. The other concerns the design of computer systems possessing whatever characteristics are necessary to implement modes of operation that are, at the very least, not objectionable from a human standpoint.

Work in the first area has been in progress during the last three years, although at a low level of intensity. A few papers by Prof. Robert M. Fano and by some of his students are listed below. In addition, Prof. Fano is preparing a short monograph based on the Centennial Lectures he gave during the Spring, 1970, at the Stevens Institute of Technology.

Work in the second area consists mainly of doctoral research by Leo J. Rotenberg. He has developed a model of the protection structures and access-control mechanisms of a multi-access computer system capable of preventing unauthorized releases of information. The model includes spheres of protection constructed out of abilities to reference programs and data segments. Processes can make calls and return from sphere to sphere through inter-sphere links. It can be shown that, under appropriate conditions, calling spheres cannot spy on their callees, nor the callees on their callers. The model includes also facilities for keeping records of critical actions (by system programmers, for instance) and for allocating responsibility for whatever a process does. Such facilities are essential to implement and enforce law regulations and contractual agreements existing in the user community. A brief summary of some of this work is presented in one of the papers listed below ("Surveillance Mechanisms in a Secure Computer Utility").

References

1. A. W. Holt and F. Commoner, Events and Conditions, Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York (1970), pp 3-52.

2. C. A. Petri, Communication With Automata, Supplement 1 to Technical Report RADC-TR-65-377, Vol. 1, Griffiss Air Force Base, New York 1966. [Originally published in German: Kommunikation mit Automaten, University of Bonn, 1962.]

3. H. J. Genrich, Simple Nonsequential Processes, Gesellschaft fur Mathematik und Datenverarbeitung, Bonn, 1971.

4. A. W. Holt and F. Commoner, Events and Conditions, Part 2, Applied Data Research, Inc., New York, N. Y.

5. S. S. Patil, Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes, Computation Structures Group Memo 57, Project MAC, M.I.T., Cambridge, Mass., February 1971.

6. E. W. Dijkstra, Co-operating Sequential Processes, Programming Languages, F. Genuys, Ed., Academic Press, New York, 1968.

7. D. E. Muller, Asynchronous Logics and Application to Information Processing, Switching Theory in Space Technology, Stanford University Press, Stanford, California, 1963.

8. J. B. Dennis and S. S. Patil, Speed Independent Asynchronous Circuits, Proceedings of the Fourth Hawaii International Conference on System Sciences, 1971.

9. W. W. Plummer, Asynchronous Arbiters, Computation Structures Group Memo 56, Project MAC, M.I.T., Cambridge, Mass., February 1971.

10. J. B. Dennis, Programming Generality, Parallelism and Computer Architecture, Information Processing 68, North-Holland, Amsterdam 1969, pp 484-492.

11. J. B. Dennis, Future Trends in Time Sharing Systems, Time-Sharing Innovation for Operations Research and Decision-Making, Washington Operations Research Council, 1969, pp 229-235.

12. J. L. Gertz, Hierarchical Associative Memories for Parallel Computation, Report MAC-TR-69, Project MAC, M.I.T., Cambridge, Mass, June 1970.

13. P. Lucas and K. Walk, On the Formal Description of PL/I, Annual Review in Automatic Programming, Vol. 6, Part 3, Pergamon Press 1969, pp 105-182.

14. J. McCarthy, A Formal Description of a Subset of Algol, Formal Language Description Languages for Computer Programming, North-Holland, Amsterdam, 1966, pp 1-12.

References (cont.)

15. P. J. Landin, The Mechanical Evaluation of Expressions, The Computer Journal, Vol. 6, No. 4 (January 1964), pp. 308-320.

16. J. E. Rodriguez, A Graph Model for Parallel Computations, Report MAC-TR-64, Project MAC, M.I.T., Cambridge, Mass., September 1969.

17. S. S. Patil, Closure Properties of Interconnections of Determinate System, Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York, 1970, pp. 107-116.

18. J. B. Dennis, Coroutines and Parallel Computation, Princeton Conference on Information Sciences and Systems, Princeton, N.J., March 1971.

19. J. B. Dennis, On the Design and Specification of a Common Base Language, Proceedings of a Symposium on Computers and Automata, Polytechnic Institute of Brooklyn. To be published.

20. I. Campbell-Grant, "The Controlled Execution of Parallel Programs Operating on Structured Data", S.M. Thesis, Dept. of Electrical Engineering, January 1971.

Publications 1970-1971

Campbell-Grant, I., "The Controlled Execution of Parallel Programs Operating on Structured Data", S.M. Thesis, Dept. of Electrical Engineering, January 1971.

Dennis, J. B., Coroutines and Parallel Computation, Princeton Conference on Information Sciences and Systems, Princeton, N. J., March 1971.

Dennis, J.B., On the Design and Specification of a Common Base Language, Proceedings of a Symposium on Computers and Automata, Polytechnic Institute of Brooklyn. To be published.

Dennis, J. B., and Patil, S. S., Speed Independent Asynchronous Circuits, Proceedings of the Fourth Hawaii International Conference on System Sciences, 1971.

Fano, R. M., "Computers in Human Society -- For Good or Ill?", Technology Review, March 1970, pp. 25-31.

Publications (cont.)

Fano, R. M., "Computers in Society", to be published in the Proceedings of the Symposium "L'Informatica, La Cultura e La Societa Italiana", held at the Fondazione Giovanni Agnelli, Torino, Italy, December 9-11, 1970.

Patil, S. S., Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes, Computation Structures Group Memo 57, Project MAC, M.I.T., Cambridge, Mass., February 1971.

Plummer, W.W., Asynchronous Arbiters, Computation Structures Group Memo 56, Project MAC, M.I.T., Cambridge, Mass., February 1971.

Rotenberg, Leo J., "Surveillance Mechanisms in a Secure Computer Utility", Computers and Society, Vol. 2, No. 1, April 1971, ACM Special Interest Group on Computers and Society.

Vogt, Carla, "Making Computerized Knowledge Safe for People", Technology Review, March 1970, pp. 33-39.

COMPUTER SYSTEMS RESEARCH

Prof. F. J. Corbató

Academic Staff

Prof. J. H. Saltzer

Instructors, Research Associates, Research Assistants and Others

D. D. Clark
J. Coffman
K. T. Pogran
R. R. Schell
M. D. Schroeder
A. Sekino

Undergraduate Students

P. B. Bishop
D. Bricklin
B. Carlson
J. R. Cone
R. Frankston
P. A. Green
R. H. Gumpertz
P. A. Karger
R. S. Lamson
M. Liu
D. Misunas
D. P. Reed
K. Rhoades
J. Stern
J. M. Strayhorn
C. D. Tavares

DSR Staff

R. H. Campbell
R. C. Daley
S. D. Dunten
R. J. Feiertag
R. L. Gardner
C. C. Garman
R. K. Kanodia
R. F. Mabee
E. W. Meyer, Jr.
N. I. Morris
M. A. Padlipsky
T. P. Skinner
M. J. Spier
V. L. Voydock
M. B. Weaver
S. H. Webber

Support Staff

O. D. Carey
S. Daise
C. P. Doyle
L. J. Haron
D. L. Jones
T. H. Seymour
A. G. Testa
M. W. Webber

Guest

Prof. G. Iazeolla

V. COMPUTER SYSTEMS RESEARCH

A. Introduction

The Computer Systems Research Group concentrates upon discovering ways to make engineering of complex information systems more methodical. Its approach is to use the Multics system as a laboratory. Thus, the work of the group must be classed as experimental, in contrast to the more theoretical attack followed by the Computation Structures Group. Use of an operating computer utility as a laboratory has both advantages and difficulties. The chief advantage is contact with reality and testing of new engineering ideas in a real operational environment, a test which is essential to achieve credibility for the ideas. The chief drawback lies in the unwillingness of live users to submit to arbitrary changes to their operating environment as a research group tries out ideas, not all of which are necessarily good ones.

Taken in proper balance, these two considerations can lead to use of a live system as a laboratory, in which a substantial number of good research problems can be adequately attacked, by careful planning. It is in such a laboratory that the group operates.

In the last twelve months, research progress has been made in several areas:

Dynamic Reconfiguration
I/O Programming Language
Automatically Managed Multilevel Memory
Protection of Programs and Data
System Programming Language
Message Handling

Each of these areas will be discussed in turn.

B. Dynamic Reconfiguration

If the "computer utility" is ever to become as much of a reality as the electric power utility or the telephone communication service, its continued operation must not be dependent upon any single physical component, since individual components will eventually fail. This observation leads an electric power utility to provide procedures whereby an idle generator may be dynamically added to the utility's generating capacity while another is removed for maintenance, all without any disruption of service to customers. A similar scenario has long been proposed for multiprocessor, multimemory computer systems, in which one would dynamically switch processors and memory boxes in and out of the operating configuration as needed. Unfortunately, though there have been demonstrated a few "special purpose" designs, it has not been apparent how to provide for such operations in a general purpose system. In a doctoral thesis done in the CSR Group, Roger R. Schell proposed a general model for the dynamic binding and unbinding of computation and memory structures to and from ongoing computa-

tions. Using this model as a basis, he also proposed a specific implementation of his model for a typical multiprocessor, multi-memory computing system. One of the results of this work was the addition to the operating Multics system of the capability of dynamically adding and removing central processors and memory boxes. The usefulness of the idea may be gauged by observing that five to ten such reconfigurations are now performed in a typical 24-hour operating day.

The full impact of this piece of research should be felt far beyond the Multics system, since the thesis provides a general model for such operations, and it can provide the designer of a new system with the insight needed to allow him to include dynamic reconfiguration in his engineering plans.

C. I/O Programming Language

An area of computer programming which has received too little attention is that of languages for specifying the detailed control of input and output devices. In most cases, the programmer expresses such control in dynamically constructed channel instruction sequences, for which his programming tools are very meager. Often, the nature of a channel program is hidden in the code of the CPU program which constructs it. Worse, the construction is usually in terms of the individual bit string constants which happen to constitute operation codes, addresses, or control messages for the channel. Thus, although the programmer may control the CPU with expressions in the PL/I language, he often controls the I/O channel with expressions in binary.

Efforts to make progress in this area are frustrating, since the nature of I/O control is very different for different kinds of devices. However, there is one class of device within which I/O control is fairly well constrained -- the class of typewriter terminals. Thus, as an experiment, a simple language was devised which permits quick and easy specification of the channel programs used for typewriter terminals. The language includes primitives for synchronization between the I/O channel and the CPU program. A translator for the language was constructed, and the Multics typewriter control package was rewritten using the language for all I/O channel control. The new typewriter control package handles all I/O with Model 33, 35, and 37 teletypes, IBM 2741 and 1050 terminals, and also static display consoles. So far, the new control package has proven much more maintainable than earlier designs, thus providing some basis for continued experimentation with I/O channel control languages.

D. Automatically Managed Multilevel Memory

By now, it has become accepted lore in the computer system field that use of automatic management algorithms for memory systems, constructed of several levels with different access times, can provide a significant simplification of programming effort. Examples of such automatic management strategies include the buffer memories of the IBM 370 models 155, 165, and

195, and the demand paging virtual memories of Multics, IBM's CP-67, and the Michigan Terminal System. Unfortunately, behind the mask of acceptance hides a worrisome lack of knowledge about how to engineer a multilevel memory system with appropriate algorithms which are matched to the load and hardware characteristics. One of the projects of the CSR Group is to instrument and experiment with the multilevel memory system of Multics, in order to learn better how to predict in advance the performance of proposed, new, automatically-managed, multilevel-memory-systems. Several specific aspects of this goal have been explored recently:

1. A strategy to treat core memory, drum, and disk as a true three-level memory system has been proposed, including a "least-recently-used" algorithm for moving things from drum to disk. Such an algorithm is already in use to determine which pages should be removed from core memory. The dynamics of interaction among two such algorithms operating at different levels are not understood, and some experimental work should provide much insight. The proposed strategy will be implemented, and then compared with a simpler strategy which never moves things from drum to disk, but instead makes educated "guesses" as to which device is most appropriate for the permanent residence of a given page. If the automatic algorithm is at least as good as the older, static one, it would represent an improvement in over-all design by itself, since it would automatically track changes in user behavior, while the static algorithm requires constant attention as to the validity of its guesses.

2. A scheme to permit experimentation with predictive paging algorithms was devised. The scheme provides for each process a list of pages to be preloaded whenever the process is run, and a second list to be immediately purged whenever the process stops. The updating of these lists is controlled by a decision table exercised every time the process stops running.

3. A series of hardware measurements were made to establish the effectiveness of a small associative memory used to hold recently accessed page descriptors. These measurements established a profile of hit ratio (probability of finding a page descriptor in the associative memory) versus associative memory size which should be very useful to the designers of virtual memory systems.

4. A set of models, both analytic and simulation, were constructed to try to understand the behavior of a shared virtual memory. The most important result of this line of work so far has been finding that a single parameter of load (the mean execution time between "missing" pages in the virtual memory) suffices to provide a quite accurate prediction of paging and idle overheads. A doctoral thesis is in progress on this topic.

As a sidelight, the measurements used to validate the models led to the discovery of an overloaded disk paging channel, the addition of a second hardware channel, and the invention of an

ingenious algorithm to maximize the effective capacity of the two channels. In brief, the two channels both connect to three hardware disk controllers, each of which can process only one request at a time. The heart of the algorithm is, when a channel comes available, to look ahead in the queue of work for the first outstanding request which is directed to one of the two unused disk controllers. Although some requests are thereby processed out of order, the over-all multiprogramming performance is improved, since the average queuing for disk service is reduced.

E. Protection of Programs and Data

A long-standing objective of the CSR Group has been to provide facilities for the protection of executing programs from one another, so that users of a public computer utility may, with confidence, place appropriate control on the release of their private information. In 1967, a scheme was proposed which provided a generalization of the usual supervisor-user protection relationship. This scheme, called "rings of protection", provides user-written subsystems with the same protection from other users that the supervisor has, yet without requiring that the user-written subsystem be incorporated into the supervisor. This scheme was brought under intense review in the last year, with two results:

1. A hardware architecture which implements the scheme was proposed. One of the chief features of the proposed architecture is that subroutine calls from one protection ring to another use exactly the same mechanisms as do subroutine calls among procedures within a protection area. The proposal appears sufficiently promising that it was included in the specifications for the next generation of hardware to be used for Multics.

2. As an experiment in the feasibility of a multilayered supervisor, several supervisor procedures which required protection, but not all supervisor privileges, were moved into a ring of protection intermediate between the users and the main supervisor. The success of this experiment established that such layering is a practical way of reducing the quantity of supervisor code which must be given all privileges.

Both of these results are viewed as steps toward first, a more complete exploitation and understanding of rings of protection, and later, a less constrained "domain of protection" organization. Two doctoral theses are underway in this area.

F. System Programming Languages

Another technique of system engineering methodology being explored by the CSR Group is that of higher level programming language for system implementation. The initial step in this direction (which proved later to be a very big step) was the choice of the PL/I language for the implementation of Multics. By now, Multics offers an excellent case study in the viability of this concept. Not only has the cost of using a higher level

language been proven acceptable, but increased maintainability of software has permitted much more rapid evolution of the system in response to research proposals as well as user needs.

During the year, progress was made on several specific aspects of exploring higher level languages:

1. The transition from an early PL/I subset compiler to a newer compiler which handles almost the entire language was completed. This transition was carried out with performance improvement in practically every module converted. The significance of the transition is the demonstration that it is not necessary to narrow one's sights to a "simple" language for system programming. If the language is thoroughly understood, even a language as complex as the full PL/I can be effectively used.

2. Notwithstanding the observation just made, the time required to implement a full PL/I compiler is still too great for many situations in which the compiler implementation cannot be started far enough in advance of system coding. For this reason, there is considerable interest in defining a subset language which is easily compilable, yet retains the features most important for system implementation. Such a language was defined, and a report has been prepared describing it.

3. An implication of using higher-level languages for system programming is that programmers find it more convenient to construct large subparts of an operating system out of many small modules. This modularity generally improves the structural organization of the subsystem by making its various functions distinct. However, when there are many modules to be assembled into a subsystem, the assembly itself requires a language to specify many otherwise tedious details of the binding which is to occur. (For example, when several procedures are bound together, usually only a few of the total set of entry points are to remain as entry points from outside the bound subsystem. Some method is needed to identify which entry points remain.) The interface between the compiler and the binder is only beginning to be understood, as another iteration of the binding specification language design was completed. One of the results of this work has been the definition of a virtual machine interface which can be respected by the compiler and the binder, but which does not exercise all of the flexibility implied by the real machine. It will take considerably more experimentation and study to determine if a real machine could be significantly simplified by removing the unused flexibility.

4. A census of Multics system modules was undertaken, to learn exactly how much of the system was actually coded in PL/I, and reasons for use of other languages. Roughly, of the 1200 system modules, about 1000 were written in PL/I, and 200 in machine language. About half of the 200 machine language modules were support routines for the early PL/I compiler providing, for example, string concatenation subroutines. Many of the rest represented tiny subroutines to execute this or that privileged instruction, etc. (No attempt

was made to provide PL/I built-in functions for every conceivable hardware need.) Significantly, only a half dozen modules (the traffic controller, the central page fault path, and interrupt handlers) which were originally written in PL/I have been recoded in machine language for reasons of squeezing the utmost in performance. Several programs, originally machine language, have been recoded in PL/I to increase their maintainability.

5. Research in techniques of compiling complex languages was continued,* with a major result being a separation of the code generation phase which is sufficient to allow the same code generator to be used for both PL/I and FORTRAN. Also, new code optimization strategies were explored.

G. Message Handling

The observation that Multics contained a large number of independent mechanisms, all of which were solving different versions of the same problem, led to a proposal for general supervisor primitives for queuing messages. It would appear that although one can in principle construct message queues in addressable memory, proper protection of previously posted messages requires a protection capability not expressable in terms of access to addressable memory. Thus the function of providing protected mailboxes for messages seems to be a primitive one, which must be provided by the supervisor or the hardware. Although message queues by themselves have been proposed and implemented in other systems, the trick is to embed them in the architecture in a natural way. In the design developed here, message queues fit into the general operating system structure in a way similar to segments. That is, they are catalogued with distinct names, and they appear in an address space as objects which act as FIFO queues. In the long run, a message queue mechamism may be an appropriate object for direct hardware implementation. To explore this area, a software message queue mechanism was designed, and added to the Multics system, and the various independent mechanisms are being scrapped.

H. Graphics Support

The CSR Group does not carry out research on techniques of graphical display. However, there are many very interesting and sophisticated ideas in the field of graphics, invented elsewhere, which have not received a true test of usefulness because they were implemented within some special purpose system. The CSR Group is attempting to integrate some of these ideas into a Multics graphics system, in an attempt to show the feasibility of making sophisticated graphical display a property of the general purpose computer utility. To this end, several development lines are in progress:

*This work was actually carried out by our Honeywell counterparts in the joint study.

1. An initial, reasonably simple, graphics display system has been designed and implemented. Its purpose was principally to test certain strategies of coupling graphics to the virtual memory, and to gain some experience in graphics.

2. Attachment and use of the ARDS (Advanced Reactive Display Station) storage tube display was accomplished.

3. Design was started on a more sophisticated graphics system which would incorporate many of the test ideas developed at Lincoln Laboratory, Bell Telephone Laboratories, and the Rand Corporation. The team doing this design is also responsible for interfacing to the graphics protocol of the ARPA network, so that the completed graphics system should be very widely usable.

I. Other Activities

Several other activities, not all of which are classed as research, were carried out by the group:

1. An interpreter for the LISP language which permits an essentially unlimited workspace within the virtual memory was implemented. The effectiveness of demand paging for LISP-type applications has long been a topic of debate, and one purpose of this work is to help resolve the debate. A LISP compiler is also being constructed.

2. The exportation of already developed ideas was pursued in a variety of ways. In January 1971, a symposium to discuss Multics was held at M.I.T., drawing about 90 attendees from industry and government. The users' manuals of the system were upgraded, and a number of technical papers were prepared and presented. A book by Elliott Organick, describing the Multics system, was accepted for publication by the M.I.T. Press, and is scheduled for Spring 1972 publication. Finally, the operating Multics system itself was exported to two other sites, the Rome Air Development Center and Honeywell Information Systems, Waltham, Mass., technical computing center.

3. In what amounts to a tour-de-force of focusing many ideas into one mechanism, a complete PL/I source language program debugging system was designed and implemented. This system, which required cooperative modifications of the PL/I compiler, allows methodical exercise of essentially every feature which a programmer might use in the segmented virtual memory environment of Multics. Many previous systems have established the value of powerful program debugging tools in an interactive environment, but most have been designed primarily for the sophisticated machine language programmer.

4. As mentioned in the introduction, the privilege of using a live system as a research laboratory is paid for partly by the necessity of being responsive to needs of a user community; a variety of tasks in this area were completed. A facility for submission of absentee jobs to the system was installed. System down time following a crash was reduced from 25 minutes

to 5 minutes. Hardware and software were modified to permit packing of page tables, to improve performance. A subsystem which permits use of the entire Dartmouth 635 time-sharing system within Multics was implemented. A better, faster, text editor based on "QED" from the SDS 940 time-sharing system was developed. Finally, at Honeywell, design was completed for an interpreter for the "APL" language of Iverson.

5. A subgroup of the CSR Group devoted most of its energy to attaching the ARPA network to Multics. This activity is reported in more detail elsewhere.

J. Acceptance of Multics

Since the earliest proposals for the creation of the Multics system, there has been a healthy skepticism expressed by many observers that a system with so many ambitious objectives could be engineered with acceptable economic performance. During the year, impressive evidence that the skeptics are wrong was amassed:

1. Use of the system by people outside the Multics development group steadily climbed to the point that 2-CPU operation during the peak hours became necessary. Even the 2-CPU system now operates at capacity for several hours per day.

2. The M.I.T. Information Processing Center, which operates the system, found that revenue from paying customers crossed the break-even point, and began to repay the initial service underwriting investments made by M.I.T. and Honeywell.

3. Revenue from paying customers exceeded that of each of the other three major computer systems at M.I.T. (these are a 360/65 running OS/MVT, a 360/67 running CP/67 and a 7094 running CTSS).

4. In addition to the 700 registered users, some 700 students used the Multics system in an "Educational Information Service" which provides a restricted service administered by a student committee.

5. A number of computer science subjects, including the introductory programming subject, found Multics sufficiently economical to use for at least part of their required home problems.

6. Enough long-term interest in Multics was apparent that the M.I.T. Information Processing Center requested from Honeywell a quotation for price and delivery of a new hardware base for the system. At the close of the reporting period, engineering specifications were largely completed, tentative schedules for delivery were worked out, and final contract details were under negotiation.

Thus, during the year Multics moved from a position of tentative acceptance to that of being the primary time-sharing service of the M.I.T. community. Initial estimates of the

price and performance of the proposed follow-on system suggest that questions of the economic viability of such a system need no longer be of concern.

K. ARPA Network Status

1. Design Issues

The technical context of the ARPA Network was described in last year's report. At that time, it was anticipated that consensus would shortly be reached among the Network Working Group participants (representing the 15 to 20 sites that will be linked by the Network) on final designs for a "Host-to-Host" (or Network Control) protocol and a "Logger" protocol (to allow direct logins over the Network to the operating systems at the various sites). However, the combined effects of the technical diversity of the systems involved and the inherent difficulty of multi-organizational design work (particularly when the sites are widely scattered geographically) resulted in a less clear-cut situation than was hoped for. By the time of the 1971 SJCC meeting mentioned earlier, neither protocol had been formally enunciated although the technical content of the forthcoming documents was sufficiently agreed upon to enable implementation to proceed. (Indeed, the Logger protocol had been split into two areas, one covering initial connections and the other covering Teletypewriter issues.) A considerable portion of the Group's directly Network-related effort during the reporting period, then, was necessarily more concerned with participation in the design process than had been supposed last year.

a) NCP: -- The protocol for the Network Control Program which each Network "host" system must implement was found to need revision after publication of its formal statement in the Summer of 1970. An important change introduced had to do with the association of byte sizes with connections and byte counts with messages. This step will be useful for allowing the "Terminal IMP" to access the Network. (A Terminal IMP is a special Interface Message Processor designed to be used alone -- not, that is, in conjunction with a large-scale local host system. Thus, the Network will be available to a much broader community of users. This is a very important aspect of the Network, in that general resource-sharing is the Network's major goal, and communication beyond the confines of host sites is a particularly desirable corollary.)

b) ICP: -- The first part of the projected Logger protocol, as noted last year, had to do with getting the attention of the remote ("server") system from the local ("user") system. This aspect has been split off into a separate "Initial Connection" protocol (ICP). Each server site agrees to listen to a designated socket and route any activity on it to whichever appropriate process, local to it, that will manage Network logins. When a request for connection arrives on the "Logger" socket, by convention a message is sent to the user designating the number of a socket over which the login negotiations are to proceed. (Actually, the designated socket and the next -- consecutively numbered -- socket are used, as connections are de-

fined to be over socket pairs.)

c) Telnet: -- The second separable aspect of the previously conceived Logger protocol was recognized to be that of common conventions for Teletypewriter use by logged-in users. A "Telnet" protocol was evolved to satisfy the perceived need for a Network-wide "virtual terminal". Again, nearly formal specifications were settled on at the Spring Network Group meeting. By convention, the Telnet processor (which may be either a separate process or simply a subroutine, depending upon particular host systems' organizations) will be invoked by the Logger at the server's site and by whatever routine manages the ICP at the user's site.

d) "Network" Command: -- A local-to-Multics command for such tasks as outgoing ICP management and Telnet invocation was also designed and initially implemented during the reporting period. Given the server system's name, it will execute the ICP and, if successful, direct user input and output over the Network, through a Telnet subroutine. It is also extensible for management of file transfer operations and other "indirect" use of remote systems (as well as the "direct" use represented by a login on a remote system) as such protocols are specified. The network command is scheduled for considerable upgrading during Summer 1971, for its role as the prime Multics user interface to the Network.

2. Implementation

Toward the end of the reporting period, an intensive effort was being mounted to complete the implementation of the above-mentioned designs in conformance with ARPA's 1 July target date.

a) INCP and Version I IMP DIM: -- Prior to the emergence of the NCP redesign, implementation had proceeded on the "Version I" protocol. The software to manage the IMP (including buffering and process wakeups, as well as link allocation and physical message formatting) was successfully installed in the Standard Service System. An Interim Network Control Program (INCP) was also implemented. The INCP, in the interests of allowing early experiments to be performed, and in order to facilitate checkout of the lower levels of software and hardware, did not implement the Host-to-Host control message aspects of the full NCP. It did, however, furnish the environment for the experiments described below.

b) Version II IMP DIM: -- By late May 1971, a revised IMP DIM which implements the Version II NCP protocol had been successfully checked out on the Development Machine and was submitted for Standard Service System installation. The installation took place in early June.

c) NCP: -- The full-scale, Version II Network Control Program being readied for installation possesses certain technical interest that is worth noting. It is essentially a finite state machine, managing sockets on the basis of state table guided transitions. This approach is expected to facilitate

dealing with future versions of the protocol. The NCP will also take advantage of the expansion of the Multics protection ring structure, residing in Ring 1 rather than Ring 0. Thus, alterations to the NCP will not require the production of new Multics System Tapes. (The IMP DIM, on the other hand, does reside in Ring 0, since it must deal with wired-down buffers. Therefore, the situation mentioned earlier in regard to the Development Machine's availability does have an impact on testing in this area.) By the end of the reporting period, the NCP was functioning successfully in the Ring 4 (user ring) environment, and awaiting Ring 1 installation.

d) Logger: -- The Multics process that will respond to the Initial Connection protocol is the standard Answering Service process. Taking advantage of the fact that the Answering Service was designed to allow various types of terminals to be connected through a common interface, the Logger implementation adds to the Answering Service code which employs an existing transfer vector-like arrangement to attach the Network I/O streams to the standard Multics process "user i/o" streams. With the I/O streams suitably attached, the processing of the login may then proceed in the same fashion as a locally initiated login. By the end of the reporting period, the Logger was functioning in the Development Machine environment, but proved to require revision before being submitted for Service System installation scheduled for late July 1971.

e) Telnet and the Initial Connection Protocol: -- Although the design considerations involving the Telnet and Initial Connection protocols were complex, implementations are relatively straightforward. The network command, which exercises both protocols, was made ready in a "stripped down" form for integration with the other Multics Network modules as they went into final testing. The test version of the network command was used to perform the Multics-to-Multics login and the logins to remote systems mentioned earlier. It will be brought up to Standard Service System quality and installed by early August.

3. Experiments

As a combination confidence test and checkout exercise of the then-current Network implementations on the respective systems, members of the Project MAC Computer Systems Research and Dynamic Modeling/Computer Graphics Groups performed an interesting experiment in December 1970. Although it employed specifically tailored processes on each system, rather than the general-purpose mechanisms envisioned for the full-fledged Network, the experiment was of interest both as a demonstration of the utility of a large fraction of the underlying machinery of the full Network and as the first in a planned series of progressively more-sophisticated experiments which take advantage of the fact that Project MAC has two Network hosts on site, with actively cooperating staffs.

The experiment involved a logged-in user on the Dynamic Modeling/Computer Graphics Group's ITS system communicating via his console with a logged-in user on Multics. ("Communicating" is

used in the sense of sending and receiving extemporaneous messages.) It was called a "polite conversation" owing to certain constraints which were imposed in order to make the experiment straightforward to implement: the conversation begins on a "speak only when you're spoken to" basis, and subsequently the participants may not interrupt when the other is "speaking". To further simplify the impelementation, the conversation was performed over an agreed-on link, with conscious catering to the respective systems' end-of-line conventions. For all its apparent triviality, the success of the polite-conversation experiment demonstrated the successful functioning of all the items then implemented. (With a change of site number, of course, it could as well have been performed across the country as merely across the building.)

The participants were so pleased with the success of the polite-conversation experiment that they decided to improvise a follow-on experiment on the spot. This entailed rerouting the ITS I/O streams to the "user_i/o" streams in the cooperating Multics process. Although the resulting "pseudo-login" quickly encountered difficulties, stemming from the line-at-a-time orientation of the polite conversation, several issues were exposed which proved to be quite fruitful in subsequent contributions to the Telnet protocol design. The polite conversation was re-enacted when the INCP and IMP DIM had been installed on the Service Machine, but it was decided not to pursue login issues until a higher degree of Network-wide consensus was reached on the protocols.

Another experiment employing current implementations was performed during the reporting period, involving the transfer of files from ITS to Multics.

Publications 1970-1971

Saltzer, J. H. and J. W. Gintell,*"The Instrumentation of Multics", Communications of the ACM, Vol. 13, No. 8, August 1970, pp. 495-500. This paper dealt with the desirability of performance metering and described various Multics performance metering tools.

Clark, D., R. M. Graham, J. H. Saltzer and M. D. Schroeder, "The Classroom Information and Computing Service", MAC TR-80, January 1971, AD-717-857. This report described an operating system designed for use in the M.I.T. Course 6.233, "Information Systems"; the system constitutes a simplified subset of Multics, and its implementation gave rise to many of the ideas proposed for the Multics follow-on hardware.

Saltzer, J. H. and J. Ossanna,* "Technical and Human Engineering Problems in Connecting Terminals to a Time-Sharing System", Proceedings of the AFIPS Fall Joint Computer Conference, Vol. 37, 1970, pp. 355-362.

*Non-MAC author.

Publications 1970-1971 (cont.)

Schroeder, M. D., "Performance of the GE-645 Associative Memory while Multics is in Operation", Proceedings of the ACM SIGOPS Workshop on System Performance Evaluation, April 1971, Harvard University, pp. 227-245.

Schell, R. R., "Dynamic Reconfiguration in Multics", Ph.D. Thesis, Department of Electrical Engineering, June 1971, also MAC TR-86, AD-725-859.

DYNAMIC MODELING/COMPUTER GRAPHICS/COMPUTER NETWORKS

Prof. J. C. R. Licklider

Instructors, Research Associates, Research Assistants and Others

R. D. Bressler
R. H. Bryan
R. J. Fleischer
F. E. Guertin
J. F. Haverty
R. Johnston
D. T. McDonald
P. L. Miller
H. F. Okrent
G. F. Pfister
W. G. Shaw
J. R. Sloan

Undergraduate Students

W. F. Bauer
A. J. Baum
E. H. Black
H. R. Brodie
M. S. Broos
K. M. Brown
A. Y. Chan
R. G. Curley
S. E. Cutler
B. K. Daniels
R. A. Freedman
D. E. Geer, Jr.
R. A. Guida
J. H. Harris
P. W. Hughett
W. F. Hui
E. Kant
E. I. Katz
R. M. Katz
R. N. King
P. B. Kurnik
P. J. Leach
P. D. Lebling
C. K. Leung
R. T. Lindsay
S. G. Morton
P. A. Pangaro
G. Pavel
R. L. Prakken
M. A. Rondio
L. M. Rubin
N. D. Ryan
H. E. Tucker
B. J. Zak

DSR Staff

B. J. Bailin
A. K. Bhushan
G. R. S. Bingham
A. L. Brown
M. A. Cohen
D. G. Cressey
S. W. Galley
R. P. Goldberg
J. P. Jarvis, III
K. J. Martin
R. M. Metcalfe
J. C. Michener
S. G. Peltan
C. L. Reeve
J. R. Taggart
A. Vezza

Support Staff

M. A. Bizot
M. F. Brescia
J. C. Cheney
M. T. Cheney
M. S. Draper
C. T. Falls
M. J. Grano
J. A. Haley
A. J. Hicks
R. F. Hill
F. L. Yost

Guest

Prof. A. Fleisher

VII. DYNAMIC MODELING, COMPUTER GRAPHICS, AND COMPUTER NETWORKS

A. Introduction

The Dynamic Modeling Group, the Computer Graphics Group, and the Computer Networks Group of Project MAC were formed last year. The efforts of the first two of those groups and about half of the third are strongly interrelated, focusing upon the design and development of a computer system specialized for highly interactive problem solving through modeling. The goals of and plans for that work were described in a section of the Annual Report for 1969-70. Progress toward the goals during the year 1970-1971 will be reported upon now in this section. The other part of the work in computer networks, also interrelated but mainly involving members of the Computer System Research Group, is reported upon in that group's section of this Annual Report.

The main objective of the joint research program of the two-and-a-half groups is a hardware-software computer system that will go significantly far beyond conventional time-sharing systems in facilitating the formulation, understanding, and solution of difficult problems through modeling. It is now widely recognized that the best medium in which to represent and experiment with the interdependencies within complex situations and processes is that of interactive computer programs. In such programs, interrelations that are amenable to mathematical representation can be expressed mathematically, those that are not can be simulated empirically, and all can be brought together to yield a model that, when executed in a computer, "runs" and exhibits dynamic behavior. The behavior, displayed to the modeler and his associates, reveals consequences of the facts and assumptions incorporated into the model and of their interrelation and organization. If the modeled situation or process is at all complex, most people can see "how it works" much more clearly by modeling it in programs and running, observing and experimenting with the model than by merely thinking about it or working on it with pencil and paper.

Heretofore, there have been specialized programming languages (SIMSCRIPT, GPSS, DYNAMO, SIMULA, etc.) to facilitate the preparation of computer-program models, but not specialized computer systems in which to observe and experiment with such models. Ordinary computer systems will execute the programs all right, and good time-sharing systems will to some extent facilitate their preparation, but existing systems are lacking in important dimensions. They do not provide a store room full of parts out of which to assemble models. They do not provide some of the tools required in experimenting with and modifying models. They do not provide displays through which one can quickly select and observe various aspects of the behavior of models. And they do not provide for progressive, accumulative augmentation of the sets of tools, techniques, parts, and models as the system is used in successive modeling projects. The aim of the program described here is to create a modeling system that will have strong capabilities in those dimensions as well as in the other dimensions of general-purpose interactive computing.

The Dynamic Modeling Group's part of the joint effort is mainly to create and assemble an array of tools and techniques, exclusive of graphics, that will facilitate modeling. The Computer Graphics Group's part, of course, is mainly to develop and provide the tools and techniques of graphic control and graphic display. The Computer Networks Group's part is to advance the art of networking and, in particular, the ARPA Network, in such a way as (a) to make remote resources (e.g., processing programs, data collections) available to the system, almost as though they were locally resident parts of it, and (b) to make it possible to use the system, with little degradation of service, from remote consoles.

The three groups share the work of assembling and augmenting the hardware computer facility and adapting the basic system programs to serve the needs of modeling, graphics, and networking. There has been, and there continues to be, much of this work. It often seems to be a diversion from efforts that would contribute more directly and more visibly to the development and understanding of modeling, graphics, and networks, but it is essential and innovative work involving new concepts in system organization for highly interactive and predominately graphical man-computer interaction.

Because three basic themes are involved in the joint program, it has been difficult to find an expressive name for the Dynamic Modeling/Computer Graphics/Computer Networks (PDP-10) system. Despite experimentation with permutations of "dyna", "graph", "cogni", "net", and other such combinable forms, we do not yet have a name that satisfies us. In this report we shall refer to the computer system -- hardware plus software plus extension into the ARPA Network -- simply as "the computer system" or "the system".

B. Dynamic Modeling

During the year, our ideas about the kit of tools and techniques and the system organization required for dynamic modeling took more definite shape, and good progress was made toward implementing some of them. Among the implementations having or approaching initial operational capability at the end of the year were those relating to mediation, intervention, the library of subroutines and data sets, the library of documentation, a programming language based on LISP but extended to handle diverse data types and to exploit graphics, and a word-oriented system for representing text.

1. Mediation and Intervention

"Mediation", as the term is used here, is the function of a program that interposes itself, in order to organize and facilitate their intercommunications, between other programs or between routines and the sets of data upon which they operate. The mediator of the computer system is a collection of programs called "CARE" ("CAll and REturn") prepared by Jeffery Harris, Paul Hughett, and J. C. R. Licklider. CARE is intended to operate within each process in the system. CARE is continually being

augmented. It is at present operating in some processes, and we expect to incorporate it into most user-level processes during the fall and winter.*

Much of the motivation behind mediation stems from the modeler's need to intervene in the operation of his model and experiment with it. We shall discuss intervention shortly. For the moment, merely note that the modeler cannot intervene and experiment conveniently if the parts of his model are linked together tightly, as by a compiler-loader or assembler-loader, when the model is introduced into the computer. The mediator CARE effects the linking as the program is executed. The modeler can therefore rearrange the parts without having to reload everything and begin over.

In the system, several different kinds of subroutine calls are recognized. They are graded in complexity and usefulness and also, of course, in cost. At the time of each complex call, and again at the time of each corresponding return of control, CARE interposes itself between the caller and the callee. CARE then handles several housekeeping chores, such as protecting the caller's information against disruption by the caller, and gives the user an opportunity to intervene. The times of calling and returning are, of course auspicious for intervention because, at those times, the transitory complexities of looping, dispatching, pushing, popping, and the like are momentarily suppressed and information is disposed in the computer memory in a relatively orderly way.

CARE comes into play, also, each time a processing routine creates or activates or deactivates or purges a set of data. The data-related functions of CARE are presently being further developed. At present the arrangement is as follows: When a processing routine wishes to create or activate a data set, it issues an order to CARE, naming the data set and (especially in the case of creation) providing other essential information. CARE then creates and names an empty data set or activates the named existing data set of the specified type and sets up a pointer or pointers to it. Thereafter, the processing routine operates upon the data set through the pointers -- at arm's length, as it were -- and in a sense never knows or cares exactly what data set it is processing. That is to say, the processing routine is (was) written to process data sets of a specified type, and what it does is specialized for the type but not for the particular data set created or activated.

---

*The word "process" is used here in the same sense as is understood among users of MULTICS: an organization, in a computer memory, of routines and data with which is associated certain housekeeping information, the most essential items of which are a pointer to the current or pending instruction and the boundaries of an address space. The computer system lets each user employ several or many concurrent and intercommunicating processes.

A basic problem is inherent in the fact that the programmer who prepared the processing routine did not know what data set later modelers would wish to have it process. In conventional computing, the user feeds the data into a card reader, and the processing routine processes whatever data come in. In the context of dynamic modeling, however, one assumes that there are several sets of data in the computer store, and the modeler may wish to substitute one of them for another during a single run of his model. Indeed, the modeler may wish to go back to a critical point in the run and see what difference would be made by a substitution of data. CARE handles this problem by letting the modeler interrupt the execution of the model at any call or return point and revise certain of CARE's bookkeeping tables. One table contains a list of translation tables. Each translation table associates "programmers' names" of data sets with "modelers' names" of data sets. By revising the tables, the modeler can direct processing routines upon whatever data sets he likes.

In order to intervene effectively, of course, the modeler must know where his model is in the course of its run. He can tell something about that from displays of its behavior, but (in one of its modes) CARE gives him the detailed picture by displaying, at each mediated call point and return point, the name of the subroutine that is being called or returning. In a submode of this "subroutine naming" mode, CARE pauses at each call and at each return and waits for the modeler to cause it to proceed by pressing the space bar on his keyboard. The modeler can proceed step-by-step to a critical point and then intervene.

To get CARE's attention, the modeler simply presses a predesignated key. CARE then responds to commands given in a simple command language. At present, this language is being augmented to cover the essential intervention interactions, and it is being "harmonized", insofar as possible, with the command languages of other programs in the system. In the interim, the modeler carries out most of his intervention functions through DDT (Dynamic Debugging Tool), a program we borrowed from the Artificial Intelligence Laboratory and have adapted to our system. DDT often operates as a process superior to the model process(es). CARE operates within each model process. CARE can transfer the modeler's interaction channel(s) to DDT, and DDT can transfer it (them) back -- with the state of the model preserved.

A most important function of intervention is the selection of aspects of the model's behavior for display and the assignment of aspects to display areas. Work on that function was in midcourse at the end of the year. The display part will be described in the section on Computer Graphics.

2. The Library of Subroutines and Data Sets

A basic part of the system is a memory-resident address table that will hold address and related information about every subroutine (except for subroutines of the simplest class) and every nonephemeral data set in the library. Each such subroutine or

data set, if not already in memory, will be automatically brought into memory (by DYNAL, a dynamic loader designed and implemented by Christopher Reeve) when needed. We still hope (cf. last year's report) to amass a library collection of at least 1,000 generally useful subroutines and 100 generally useful data sets. Our progress toward the subroutine part of that goal has seemed fairly rapid when judged without reference to interrelation and documentation. Members of the group are prolific programmers and excellent debuggers. When judged in terms of the subroutines fitting into the system, working with one another, and being comprehensible to users, however, the situation at the end of the year was disappointing. A major campaign to get the library organized and documented is in the offing. The library of source programs will be maintained in a compact form by ARCHIV, a file-compressing program designed and implemented by Allen Brown, Robert Bressler, John Haverty and Christopher Reeve. The corresponding object programs will be maintained by ATTACH, designed and implemented by Bruce Daniels. Both ARCHIV and ATTACH are now operating.

3. The Library of Documents

We have had operational on MULTICS, for about a year, a computer-based system, developed by Richard Bryan, for storing and retrieving information about the software of the modeling system. At present, that system is being augmented and interfaced to the PDP-10 via the ARPANET. The corpus of documentation includes, as of the end of the reporting year, one-page abstracts and multipage documents describing subroutines, data sets, macros, and the like.

The main accomplishment of the year, insofar as documentation is concerned, was the formulation and description of the set of standards for program preparation and documentation called "Convention II". Convention II is described in a series of 20 documents by David Burmaster, Martha Draper, Paul Hughett, Karolyn Martin, J. C. R. Licklider, Christopher Reeve and Albert Vezza. Convention II deals with the various policy and technical aspects of program preparation and documentation. The topics include a standard format for documents, three standard glossaries (notation, abbreviations and expansions, and system-wide terms), the format of subroutine headers, data-set headers, and address tables; data types; the naming of files; two kinds of abstracts; organization and format of listings; the mediator and its functions; a set of system-wide macros and how to use them; and other such topics. At the end of the year, Convention II was ready for promulgation. A significant part of the ensuing effort will be to bring all of our software that "has a future" into accord with Convention II and to make it operate within the context of the mediator CARE.

One of the provisions of Convention II is that, associated with each software entity, such as a subroutine, a data set, or a macro, there must be an explanation of the mnemonics of the name, a meaningful expansion of the name, an abstract containing prescribed classes of information, and a set of descriptors. The descriptors, mnemonics, expansions, and -- in the case of entities to which it is applicable -- the calling and returning

sequence format will be available on-line through an information-retrieval system associated with CARE. The abstracts, listings, and other lengthier documents about the software will be available in a small ink-and-paper library at each console. As indicated earlier, we set up a first version of part of such a system on MULTICS last year. Now we are beginning to connect it to the PDP-10 system. Effective information retrieval is an essential part of the concept of the system we are developing.

4. An Extension of the LISP Language

Our design objective of highly interactive experimentation with models is inconsistent with the classical organization of software based on compiling and loading monolithic masses of software in which all the parts are rigidly linked together. The limitations of the classical organization have been broken in two main directions, on the one hand in the paradigm of MULTICS, in which linking is deferred until reference is actually made, during the execution of a program, to subordinate parts that should be linked to the parts already running, and on the other hand in the paradigm of LISP, in which editing, debugging, and other such activities are carried on within the coherent framework of the language implementation. For our purposes, both MULTICS and LISP have many desirable features, but neither in and of itself provides the desired facility for the kind of modeling to which we aspire. To mention the main shortcomings, MULTICS is not set up for use with a graphics processor operating out of main memory, and LISP, while highly coherent internally, is difficult to bring into relation with external software and is only weakly developed in the directions of data typology and graphical interaction. We have therefore been exploring the problem of incorporating into our system the best of the two worlds. Members of our groups have been working with the members of the Artificial Intelligence Laboratory on the design and implementation of an extension of the LISP language that will provide a number of advantages over previous versions of LISP. These will include data type checking, lexical scoping, recognition of a large number of elementary and compound data types, and the inclusion of primitives upon which to erect a graphical interaction subsystem.

The extension of LISP is known locally as "MUDDLE". It was designed and has been implemented by Carl Hewitt and Gerald Sussman of the Artificial Intelligence Laboratory and Christopher Reeve, David Cressey, Bruce Daniels, and Gregory Pfister of Dynamic Modeling/Computer Graphics/Computer Networks. MUDDLE is operational now as an interpreter. As implemented, it is rather separate and distinct from the other software of the system we are developing. Wishing to bring MUDDLE into our system in such a way as to integrate its advantages coherently, we are studying the possibility of merging MUDDLE's data types with the system's data types and MUDDLE's implementation routines with the system's subroutine library.

We hope to use MUDDLE as an interpreter for the upper levels of the subroutine-calling hierarchy. The idea is to employ interpretation in the upper levels, where most of the changes are made

in the course of exploring a model, and to employ assembled or compiled subroutines at the lower levels, where the time efficiency of execution (as opposed to interpretation) is most important. It now seems likely that such an organization of software may be feasible. On the other hand, it would not be good to cause our version of MUDDLE to diverge greatly from that of the Artificial Intelligence Laboratory because MUDDLE is envisoned as the base for the implementation of PLANNER, and PLANNER seems likely to be very useful in modeling as well as in artificial intelligence research.

5. Lexicontext

Fundamental to the design of an integrated or coherent information system is the selection of a basic informational building block. In most computer systems, as in ours at present, the basic atom of information is either a character (byte) or a computer word. For substantive modeling applications, however, the character is too small a unit, and the computer word does not bear a direct enough relation to the words of natural languages in terms of which people think. Looking toward a future in which a good computer system will have, and be able to use knowledgeably, a vocabulary of tens or hundreds of thousands of words of natural language, we have developed a system, called "Lexicontext"*, that gives to the word -- the word of natural language and/or the word of technical jargon -- the role of basic building block. In the Lexicontext system, a word is processed, not as a string of characters, but as a pointer to an argument in a lexicon of argument-function pairs. The lexical function is divided into subfunctions. The absolutely essential subfunction is the spelling of the word. Other subfunctions can be added with apparatus provided by Lexicontext. They will include additional morphological information, syntactic information, synonyms, definitions, and (hopefully eventually) programs that give the entries operational meaning in the paradigm of Winograd's PROGRAMMAR. Most of these subfunctions can be implemented in terms of the basic Lexicontext element, the pointer to an entry in the lexicon.

Lexicontext has been implemented by John Haverty. In his implementation, text files are composed of elements of uniform size. Each item of text (except for literals) occupies the same number of bits of storage (18 bits in half-word mode, 36 bits in full-word mode) and -- as explained -- each element represents a lexical word by pointing to its location in the lexicon. The uniformity of representation makes it convenient for the computer to process text -- e.g., to search for instances of a given word (type) or to parse sentences. The primary lexicon, itself, consists of alphanumeric strings (spellings). Associated with each primary entry there may be pointers into one or more data bases. These data bases are to contain the subfunctions, other than the spelling, mentioned earlier.

---

*It has a lexicon and deals with text, and we hope that it will provide a lexical context for work in modeling.

Actually, Lexicontext provides for 32 separate lexicons, either "new" or "old". (New lexicons can be updated on-line or off-line; old, only off-line.) The current implementation has a provision for automatic construction of a new lexicon from a conventional text file; for adding words on-line to a new lexicon, as new words, not already in any of its lexicons, arise; for merging a new lexicon into existing old lexicons, and for converting conventional text files into Lexicontext text files and vice versa. Each lexicon allows for 2^{15} elements, i.e., a vocabulary of slightly over 32,000 lexical words.

Like Hypertext[1] and NLS[2], Lexicontext structures text in a hierarchy such as volume/chapter/paragraph/sentence/word. There is a mechanism for representing extra-hierarchical items such as footnotes and references.

Lexicontext text files are more compact than character-code files. The compression ratio is not great, but it is good to gain something in compactness instead of having to trade off compactness for the efficiency of processing uniform tokens.

C. Computer Graphics

The Computer Graphics group has made good progress, during the past year, in mastering the Evans and Sutherland display subsystem and in solving basic problems in the application of graphics to facilitate human understanding of, and modeling of, complex processes and organizations. The Evans and Sutherland display subsystem (E&S) is a very powerful one, not easy to exploit fully (especially in a time-sharing environment), and much of the effort in graphics has been devoted to bring the capabilities inherent in the E&S into the hands of users who are not display specialists.

Some of the problems (as well as the advantages) of the E&S stem from the fact that it has its own processor, which operates in parallel with the PDP-10 processor and the disk channel and shares memory with them. Since memory is dynamically allocated by the operating system to the several user processes that are running concurrently in the PDP-10, it is necessary for the operating system to mediate the use of the display processor or to make corresponding adjustments of memory allocation for it. This has not been a difficult problem so far because we have been using only one E&S display. We have followed the mediation approach with success. However, we are now moving toward time-sharing the E&S display subsystem among four consoles, and that move is not trivial. Michael Brescia is designing the display time-sharing system.

1. "Picture Framing"

One of the capabilities of the E&S display subsystem is to focus its efforts mainly upon any specified small area or areas of a very large surface on which there is a picture. That capability is important because, without it, the processor would spend most of its time processing parts of the picture that lay outside the areas of interest. Accordingly, during the past year,

James Michener and other members of the Computer Graphics and Computer Networks Groups devised a technique, a kind of "picture-framing service", in which the E&S processor eliminates the extraneous parts of the picture and constructs, in memory, a sub-picture limited to a specified area or areas, and then the PDP-10 processor reformats the delimited picture for transmission to an Advanced Remote Display Station (ARDS) or an Imlac console. That technique is used as a service inside our computer system, and is being made available through the ARPANET to users of ARDS and Imlac consoles at remote locations.

2. Polyvision

Within the general context of modeling, one of the main graphics problems is display management. A modeler may have a dozen or more things to display but only a small display area -- a ten inch square or at best a very few such squares -- in which to display them. Polyvision is a display-management subsystem, designed and programmed by James Michener, Edward Black, and others, that permits the modeler to assign the various aspects of his model, mainly dynamic aspects, to named display areas and then to move the areas about, magnifying some and causing others to contract, either under program control or under the control of a stylus in the modeler's hand. Polyvision will be brought into interaction with the mediator CARE in due course, but it will not be necessary for the modeler to halt the execution of his model to adjust the configuration of his display. The modeler can control the display subsystem while the model is running. This is in line with our basic concept of graphical display as an aid to observation. It should not be necessary to build a schedule of observation into the basic framework of the model itself. The schedule of observation must be flexible and under the modeler's control throughout the course of observation.

Eventually, it may be possible to make significant changes to the model while it is running. However, the problem of modifying the model "on the fly" is more difficult than the problem of modifying the observational procedure "on the fly". The latter can be thought through and implemented once and then used with various models; the former seems, in the present state of the art, to require model-specific operation.

3. Graphical Debugging

Computer Graphics offers promise of breaking through one of the most resistant barriers to human comprehension of complex computer programs. The barrier is, figuratively speaking, the opaque integument of the computer, which deprives the observer of any global view of what is going on inside. Even with the best conventional debugging aids, such as DDT, the observer can see into the inside workings of the machine only through a very small aperture. At the operator's console, there are perhaps a few pilot lights, but they do not encode information in a very meaningful way. At a typewriter console, one can open and look inside one memory register at a time. With a graphic display, on the other hand, one can see a map of the computer memory (either in the literal space of memory registers

or in the symbolic space of source-language statements) and watch the behavior of the program on the surface of the map.

During the past year, we made some progress toward realization of a meaningful, global display of program behavior. Stuart Galley completed a graphical display of program behavior called "ESP", and Paul Hughett completed important portions of a graphical debugging tool, called "GDT". In both programs, selected registers and segments of program are represented schematically upon the display surface, and the flow of information is represented by moving arrows, moving symbols, and other devices. These programs represent only small steps toward what should eventually be a very powerful aid to the understanding of program dynamics, but they will, themselves, find useful application in the computer system.

4. Elucidations

The difficulty of harnessing a powerful display subsystem in the interest of man-computer communication about complex processes is balanced by the simplicity of getting such a subsystem to display mathematical functions. It is easy to produce all kinds of "graph paper" on the display screen, and it is easy to create all kinds of curves and surfaces. It is remarkable how much one can learn from a few minutes of play at the graphics console -- a few minutes spent in exploring mathematical functions through graphical display. Obviously, the general problem of relating graphical and symbolic representations to one another is very important in the understanding of mathematics. Obviously, a digital computer with a good graphics subsystem can greatly facilitate the development of such understanding.

During the year, several members of the laboratory developed graphics programs that provide insight into simple mathematical phenomena. These included two-dimensional and three-dimensional function plotters (Edward Black, Scott Cutler), a Fourier transformer (Robert Freedman) and a simulation of the interplay of gravitational forces in a galaxy (Paul Hughett).

5. Visual Statistical Analysis

Flowing from the general line of observation just mentioned was a major effort by Robert Fleischer called "Visual Statistics". This program brings together in a subsystem a collection of processing and display operations that facilitate visual analysis of the relations that exist within a collection of data. The operations permit the selection of data on the basis of various criteria, the plotting of the data in various modes and formats, projection from a multidimensional space to a two-dimensional surface, curve fitting, and so on. We hope to incorporate the Visual Statistics subsystem into a larger system of data-analysis routines so that we can bring both intuitive and algorithmic analysis procedures into productive interaction.

6. Imlac Displays

Although not as sophisticated as the E&S display subsystem, the Imlac consoles, which include minicomputers as well as cathode-ray-tube displays, are potentially very capable. Our use of them thus far has been largely limited to alphanumeric processing and display, but we are beginning to exploit their potential for display of graphs, charts, and diagrams.

David Lebling prepared a PDP-10 assembler for Imlac programs. Stephen Peltan prepared a loader that loads the Imlac from the PDP-10. Lawrence Rubin and Stephen Peltan developed split-screen techniques for Imlac consoles and did the programming required to make the Imlac's control keys convenient to use in our applications. And John Haverty designed and implemented a program, IMEDIT, that makes it convenient to use the Imlac editor in conjunction with the PDP-10 file-handling system. IMEDIT moves from the PDP-10 to the Imlac consoles much memory-space-consuming but trivial work. All in all, the Imlac consoles are now quite convenient and effective for applications that are limited to alphanumeric information, and they are well on their way to supporting applications that involve line drawings, also.

D. Computer Networks

The part of the Computer Networks Group to which this report pertains is the part concerned immediately with the PDP-10 computer system. Last year, the word "immediately" would not have had much significance, for the network program was just getting under way, and energies were focused mainly on getting MULTICS and the PDP-10 into communication with each other and other computers in the ARPANET. At the end of this reporting year, however, one can sit at a PDP-10 and carry out his computing operations mainly in any one of several other ARPA network machines. Most of the work to be reported upon here was aimed at creating the basis in computer communications, through computer programming in the PDP-10, for interaction between the PDP-10 and other network computers.

1. Network Control Program

Robert Bressler and other members of the Computer Network Group developed several progressively improved versions of a Network Control Program (NCP) for the PDP-10 computer. This program establishes and maintains connections between processes in the PDP-10 and other ARPANET computers. The next step was to design and implement programs that, using the NCP, would make the PDP-10 a part of an alphanumeric telecommunications network (TELNET) within the ARPANET. The TELNET programs are of two kinds, "servers" and "users". The first TELNET server program completed was a Logger, the function of which is to permit users of other network computers and users connected to the network via a TIP to log into the PDP-10 in the same way as local users. The design of the logger involved Robert Bressler, Robert Metcalfe, and Arvola Chan, and most of the programming was done by Chan.

The next program in the logical hierarchy of network software was a TELNET user program, designed and prepared by Robert Metcalfe, the function of which is to permit a user logged into the PDP-10 to log into another network computer. It handles terminal communication to and from the PDP-10, including communication with a remote TELNET server program, through the Network Control Program. Together, the TELNET server and user programs and the NCP provide the basic means of communication with remote computers and/or terminals.

Even before the basic means of communication were perfected, attention turned toward the design of yet higher levels of network software, Abhay Bhushan became interested in the protocol for the transfer of data and for the transfer of files of data. His interest led him to the chairmanship of the Committee of Data and File Transfer Protocol for the ARPANET. At the end of the reporting year, he was working on software implementations of the tentative protocols that had been thus far formulated. Members of the Computer Networks and Computer Graphics Groups turned their attention, jointly, to problems of graphical communication through the network.

Rather early in the year, interesting explorations of graphical communication through the network were made in cooperation with members of the Aiken Computation Laboratory at Harvard. Graphics programs were originated in the Harvard PDP-10 transmitted through the network to the Project MAC PDP-10, processed there by the E&S subsystem, transmitted back to a PDP-10 computer at Harvard, and displayed there on a Digital Equipment Model 340 display. The same pictures were displayed on the E&S display at M.I.T., and the characteristics of the network-mediated processing and display were compared with those of wholly local processing and display of the same program material. It was found that there was almost no perceptible difference between network and local display of single frames. In dynamic display of continuously moving pictures involving 10 to 100 lines, however, there was a big difference. The local display presented perceptibly continuous motion, whereas the network display jerked from one configuration to another 2 or 3 times per second. That was a rather preliminary test, made at an early stage of network development. Improved means will provide improved performance. We shall make further tests to determine the ability of the network's programs, Interface Message processors, and 50-kilobaud lines to handle kinematic graphics.

2. The Network at the End of the Year

At the end of the reporting year, the PDP-10 wing of the Computer Network group at Project MAC was in the process of consolidating its basic software subsystems and pressing upward into he higher echelons of the hierarchy of network software. The process of consolidation will be time-consuming because, throughout the year, the network effort was proceeding as rapidly as possible toward intermediate objectives, and the hurry to achieve them pushed aside such considerations as thorough testing and documentation. There is still some testing to be done, and there is a large amount of documentation.

At the same time, there is a keen sense of anticipation within the group, a strong motivation to master the transfer of data sets and files and to proceed as rapidly as possible to the execution in remote computers of subprograms called by programs in our PDP-10. We want to exploit network subprogram linking in order to bring functionally within the scope of our library several very useful collections of programs that exist in remote computers -- collections that we need and that would be prohibitively expensive to reprogram for the PDP-10.

References

1. Carmody, Steven, Walter Gross, Theodore H. Nelson, David Rice and Andries van Dam, A Hypertext Editing System for the /360, Pertinent Concepts in Computer Graphics, M. Faiman and J. Nievergelt (Eds.), pp. 291-329, University of Illinois Press, Urbana, 1969.

2. Engelbart, Douglas C., and William K. English, A Research Center for Augmenting Human Intellect, Proceedings 1968 Fall Joint Computer Conference, Vol. 33, part one, pp. 395-410, The Thompson Book Company, Washington, D.C., 1968.

Publications 1970-1971

Baum, Allen J., Minicomputers: Status and Architecture, Tech Engineering News, 52, No. 8, pp. 25-30, January 1971.

Black, Edward H., A Data Structure Dumper, S.B. Thesis, Dept. of Electrical Engineering, June 1971.

Bressler, Robert D., Interprocess Communication on the ARPA Computer Network, S.B. and S.M. Thesis, Dept. of Civil Engineering, June 1971.

Cutler, Scott E., Computer Graphics, Tech Engineering News, 52, No. 8, pp. 17-21, January 1971.

Goldberg, Robert P., Hardware Requirements for Virtual Machine Systems, Proceedings Fourth Hawaii International Conference on System Sciences, January 1971.

Licklider, J. C. R., Libraries and Information, reprinted from Libraries of the Future, M.I.T. Press, 1965, in The Computer Impact, 260-270, Irene Traviss (Ed.), Prentice Hall, Inc., New York 1970.

Licklider, J. C. R., Social Prospects for Information Utilities, The Information Utility and Social Choice, Sackman and Nie (Eds.) AFIPS Press, Montvale, N.J., 1970.

Publications (cont'd)

Licklider, J. C. R., The Role of Computer Graphics, The Computer Utility: Implications for Higher Education, Michael A. Duggan, Edward F. McCartan, and Manley R. Irwin (Eds.), Heath Lexington Books, Lexington, Mass., pp. 11-16, 1970

Lindsay, Robert Thomsom, Jr., A Design for a Graphical Compiler, M. I. T., S. B. Thesis, Dept. of Electrical Engineering, June 1971.

Vezza, A., and Knudson, Donald R., Remote Computer Display Terminals, Computer Handling of Graphical Information, SPSE, R. D. Murray (Ed.), July 1970.

EDUCATION

Prof. R. R. Fenichel
Prof. J. Weizenbaum

Instructors, Research Associates, Research Assistants and Others

J. Kaplan
S. A. Ward

Undergraduate Students

R. H. Brown
D. M. Christie
R. P. Silberstein

Support Staff

N. Chen
A. Garrity
M. A. Hoer
K. Young

Guests

J. Berger, Prof.
P. Eisenbert

VIII. EDUCATION

The Project MAC Progress Report V (July 1967-July 1968, p. 98 et seq.) describes the language, at the heart of the TEACH system, which then was called PL/2 but which since has been designated UNCL (UNcommonly Clean Language):

> "It is an interactive language that somewhat resembles JOSS, but differs from JOSS and other JOSS-like languages in several major respects: for example, the presence of block structure, a context editor, and a function-tracing feature".

During the year ending June 1971, the UNCL interpreter was completed. A result of this effort was design of a novel means of implementing variables of label mode.

Experiments were undertaken with a novel hardwired device which was designed to search for certain useful configurations of flip-flop circuits.

Publications 1970-1971

Fenichel, Robert R., "Design of Languages for Elementary Programming Instruction: Lessons of the Teach Project", Proceedings of IFIP Conference on Computer Education (August 1970), III, pp. 175-177.

Fenichel, Robert R., List-Tracing in Systems with Multiple Cell-Types", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation (March 1971), pp. 242-247.

Fenichel, Robert R., "Comment on Cheney's List-Compaction Algorithm", Communications of the ACM, XIV, 6 (June 1971).

Fenichel, Robert R, "On Implementation of Label Variables", Communications of the ACM, XIV, 5 (May 1971).

IMPLICIT COMPUTATION

Prof. M. L. Dertouzos

Instructors, Research Associates, Research Assistants and Others

F. G. Abramson
M. W. Dickens
M. E. Kaliski
P. A. King
M. P. Lum
C. W. Lynn
J. R. Stinger
A. S. Weinberg

Support Staff

L. E. Yaple

IX. IMPLICIT COMPUTATION

A. Introduction

This research concerns novel machines and algorithms for the solution of large-scale problems, e.g., linear and non-linear systems of equations. Two approaches are taken. In the first, the problem is separated into two parts, one of which is computed exactly (i.e., digitally with a large number of bits), and an interrelated second part which is computed inexactly (either digitally with fewer bits, or by analog means). The exact part checks whether a proposed solution is indeed a solution, while the inexact part revises the proposed solution on the basis of results of the exact computation. We have developed decompositions which compute an exact solution to linear systems of equations via iteration of the above exact-inexact computing cycles.

In the second approach, the system of equations to be solved is simulated by a spatial interconnection of computing elements. Each such element processes two types of variables called "pressures" and "flows". The flows correspond to the usual variables (including the unknowns) which are to be processed by the computing element, while the pressures signify the extent by which the flows do not satisfy the intended computation of that element. When several such computing elements are interconnected into a composite system, the pressures are used to steer the flows toward solution values. Moreover, the composite system is, by construction, of the same form as the constituent elements, i.e., it processes pressures and flows, and may therefore be used to build up larger systems. The pressures and flows are constrained by each computing element and composite system to obey a pseudo-energy constraint analogous to that obeyed by variables and co-variables of "energy lossy" physical systems. This, in turn, guarantees stability of the over-all computation.

In addition to these specific approaches, we have investigated some related fundamental issues. These concern the space-time and time-accuracy trade-offs in computation, as well as the logical capabilities of continuous (or analog) computing systems. Two basic results in this area have given rise to two papers (one by Abramson on Turing Machines for the real numbers, and one by Dertouzos on time bounds of space computations). Both papers have been accepted for presentation at the 1971 Switching and Automata Theory Conference of the IEEE.

The machines and approaches that we are investigating have applications as special-purpose computers for the rapid solution of large problems (e.g., weather forecasting, space navigation, and process control), as companions to general-purpose machines, and as algorithms that can run on conventional digital computers. These applications are briefly discussed, along with the fundamental-research activities, in the following sections.

B. Exact-Inexact Machines and Approaches

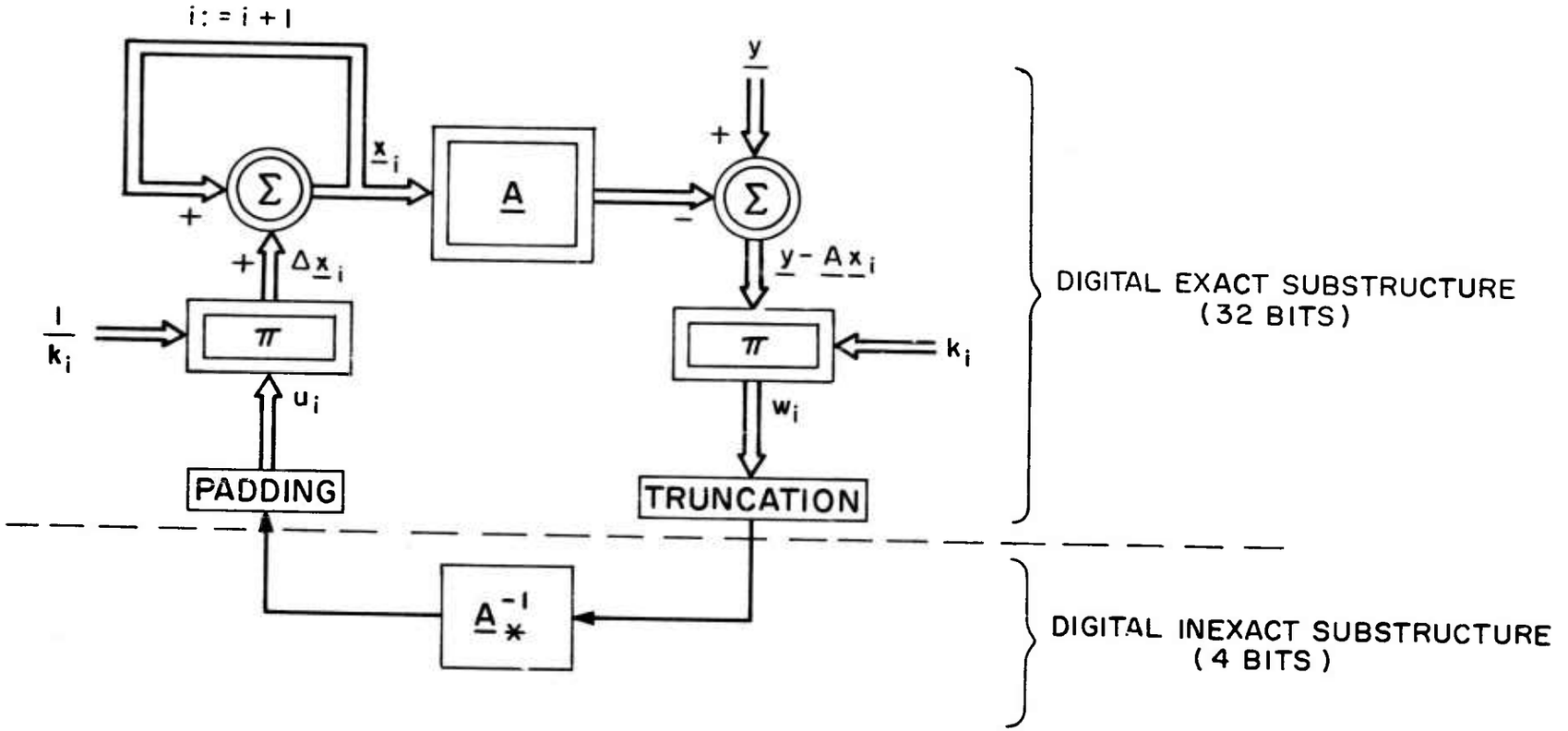
In Progress Report VII, we described exact-inexact machines with analog inexact substructures. We have investigated in considerable detail one such machine for the solution of systems of linear equations. We have found that the proposed approach is feasible and can handle fairly large problems. One limitation that we encountered is that, in the case of relatively ill-conditioned problems, the analog errors grow with increasing system size to the maximum acceptable error (for convergence of the exact-inexact cycle) of ± 50%. We are currently attempting to find exactly where that limit is. It seems to occur at system sizes of several hundred equations, for typical problems.

We have also initiated work on what seem to be promising exact-inexact approaches with a digitally computed inexact part. One such approach is presented in the remainder of this section, in terms of an example.

Consider the structure of Fig. 1. It is intended for the solution of certain systems of linear equations. Specifically, the system to be solved is $\underline{A}\underline{x}=\underline{y}$, where vector $\underline{y}$ and matrix $\underline{A}$ are given, and vector $\underline{x}$ is the unknown. For an initial explanation, let the scalar k_i shown in Fig. 1 be unity. The computing structure consists of two parts -- a relatively exact substructure (e.g., 32 bits) which checks if a suggested vector $\underline{x}_i$ satisfies the above equation (to that accuracy); this substructure computes digitally the error $\underline{y} - \underline{A}\underline{x}_i$. The other part of the system is a relatively inexact substructure (e.g., 4 bits) which computes digitally the correction variable, $\underline{\Delta x}_i$ as follows: $\underline{\Delta x}_i = \underline{A}^{*-1} (\underline{y} - \underline{A}\,\underline{x}_i)^*$, where the star subscript indicates inexact approximation (truncation) of the corresponding quantity to this reduced accuracy. Execution of the computation is iterative, each iteration consisting of first the computation of the error, then of the correction $\underline{\Delta x}_i$ and then of the next iterate $\underline{x}_{i+1}$ as $\underline{x}_{i+1} = \underline{x}_i + \underline{\Delta x}_i$. If the starred quantities were exact, then the exact solution would be obtained in one iteration, since $\underline{x}_i + \underline{\Delta x}_i$ can be easily verified as the solution of the system $\underline{A}\,\underline{x} = \underline{y}$, for any choice of $\underline{x}_i$. Because these quantities are not exact, each iteration brings $\underline{x}_{i+1}$ closer than $\underline{x}_i$ to the solution, with contraction depending on the error between their exact and inexact values.

Observe that the idea that is illustrated by this example is the decomposition of the problem into exact and inexact substructures, not the issue of stability of the above feedback approach. The stability of iterative algorithms for the solution of linear systems has been treated extensively in the literature. It is expected that the exact-inexact approach is applicable to the majority of these algorithms; indeed, a part of the proposed work is concerned with such applications.

We return now to the role of the constant k_i and of the multiplications of Fig. 1. As $\underline{x}_i$ approaches the solution, with



Σ MEANS DIGITAL EXACT SUMMATION

π MEANS DIGITAL EXACT MULTIPLICATION

⟹ MEANS EXACT INFORMATION

→ MEANS INEXACT INFORMATION

FIG. 1. EXAMPLE OF EXACT - INEXACT APPROACH.

increasing i, the error $\underline{y} - \underline{Ax}_i$ is computed exactly. It is then multiplied by k_i and converted to an inexact value by truncation of the undesirable least-significant bits. This multiplication is performed in order to increase (scale) the magnitude of the error to as near as possible the full scale of the inexact subsystem, so that the inaccuracies of that subsystem are small compared to the values of its input variables. Thus, the input to the inexact subsystem is the truncated value of the quantity $k_i(\underline{y} - \underline{Ax}_i)$. The effect of constant k_i is "cancelled", after the correction vector has been computed by the inexact subsystem, through multiplication of that vector by $1/k_i$. Naturally, the truncated correction vector is padded to the right with a sufficient number of zeros to offset truncation, i.e., to match the word length of the exact process. Thus, as computation progresses (increasing i) and the exact error $\underline{y} - \underline{Ax}_i$ becomes progressively closer to zero, the constant k_i is made progressively larger so as to keep the inexact system inputs near full scale. Observe also that these multiplication and division operations are performed to full accuracy.

Observe next that, using straightforward techniques, the exact subsystem computes the error in time proportional to N^2q^2 for an N x N matrix A, at a word length of q bits. This is the case, since the exact subsystem performs a matrix-vector multiplication.

The inexact subsystem, however, can invert the matrix in time of order N^3z^2, where z is the word length of the inexact computation. This is the case since N^3 operations are needed and, of these, multiplication is dominant, requiring time z^2. Once the matrix is inverted, the time expended per inexact cycle is N^2z^2 since an N x N matrix multiplies an N-vector, at z bits. Finally, the number of exact-inexact cycles needed is of order q/z since, at each iteration, the exact error is reduced by roughly z bits (recall that the exact error is constantly scaled up by k_i). Thus, the total time for the entire process grows as:

$$N^3z^2 + \frac{q}{z}(N^2q^2 + N^2z^2)$$

For large N, this computing time grows essentially as N^3z^2. Thus, compared to an equally straightforward exact matrix inversion approach requiring time N^3q^2, the above approach is faster by a factor $(q/z)^2$. For our example, q/z = 8, hence that factor is 64.

C. Pressure-Flow Machines

In this approach, the computing structures under consideration involve the spatial interconnection of computing elements which correspond to the individual relations (or equations) that make up the over-all problem. These computing elements, in turn, process two types of variables, which we call "flows" and "pressures".* The flows correspond to the usual variables in any computing system, i.e., the unknowns and any intermediate

*These are simply names of variables motivated by physical systems. We are not referring here to any physical pressures or flows.

variables needed to compute these unknowns. The pressures, on the other hand, denote the extent by which the flows do not satisfy the relations represented by each computing element. Each computing element treats the flows as inputs and the pressures as outputs. Thus, if the flows satisfy the intended relation of that element, then the pressures are zero. If, instead, the flows do not satisfy the intended relation, then the pressures assume non-zero values which (1) denote the extent by which the flows do not satisfy the relation, and (2) are related to the flows through a pseudo-energy constraint, similar to the relationship of variables and co-variables of physical energy-lossy elements. These, as well as certain additional properties of the pressures and flows are retained under composition of the computing elements into larger composite systems -- that is, the resultant systems have flows for inputs and pressures for outputs, which are related by the same type of pseudo-energy constraints. The result of this organization is the ability to construct arbitrarily complex, spatially distributed structures that simulate large systems of equations and that are capable of converging asynchronously to desired solutions, in the same sense that aggregates of passive electrical network elements converge on their "solutions", under given excitations.

In more detail, the organization of pressure-flow machines is as follows:

1) Primitive digital computing elements are made to correspond to the desired primitive relations. Each such element has as many inputs (flows) and as many outputs (pressures) as there are variables in the primitive relation. These pressures and flows are related through a pseudo-energy function, as follows:

a) The flows are the variables of the primitive relation.

b) The pseudo-energy function is defined on these variables, such that it is zero if and only if the values of these variables satisfy the corresponding relation. Otherwise, the pseudo-energy function is positive.

c) The pressures are defined as the gradient of the pseudo-energy, on the space of the flow variables.

2) Composite pressure-flow machines are made up of primitive computing elements, and (recursively) of composite pressure-flow machines, in direct correspondence to composite relations, which are made up of primitive relations and (recursively) of composite relations. The rules are as follows:

a) External variables, i.e., free variables of the composite relation, appear as flows and as pressures of the composite machine. As flows, they are simply connected to the constituent machines, if the

corresponding free variables are related by constituent relations. As pressures, they are generated by summation of the corresponding pressures of all constituent machines which relate that free variable.

b) Internal variables, i.e., variables bound by the composite relation, appear as neither flows nor pressures of the composite machine. Instead, each such flow is generated (negatively) by digital integration of the sum of all corresponding pressures supplied by constituent machines, i.e., the machines corresponding to constituent relations that relate that bound variable.

c) The pseudo-energy associated with a composite machine is the sum of the pseudo-energies of the constituent machines.

3) Under these composition rules, it is the case that

a) The flows of every composite machine are the variables of the corresponding composite relations.

b) The pseudo-energy of every composite machine is non-negative. In particular, it is zero if and only if the pseudo-energy of every constituent machine is also zero, i.e., if every constituent relation is satisfied, which means that the corresponding composite relation is also satisfied.

c) The pressures of the composite machine are the gradient of the pseudo-energy of the composite machine, since they are formed by addition of the pressures of constituent machines, and since the pseudo-energy of the composite machine is the sum of the constituent-machine pseudo-energies.

Observe that the properties of pressure, flow and pseudo-energy for composite machines (Items 3 (a), (b) and (c) above) are the same as the properties of the corresponding entities of primitive computing elements (Items 1 (a), (b) and (c) above). This consistency under recursion is essential, for it insures that pressure-flow machines of arbitrary complexity, constructed by the above rules, obey a fixed set of properties. These properties are, in turn, pivotal in the ability of pressure-flow machines to solve satisfactorily specific classes of problems.

One of the principal results to date is that the pseudo-energy of every composite machine decreases or at worst remains constant if the flows of that composite machine are held constant. We have further shown that for linear problems (i.e., aggregates of linear primitive relations), which are not singular, the over-all pseudo-energy decreases, converging towards the solution. These results make possible the application of the pressure-flow machines to problems of arbitrary complexity.

D. Fundamental Work

The pressure-flow approach and the inexact part of an exact-inexact machine are made up of spatially distributed systems. In order to probe the ultimate computing speed of spatially distributed systems, we have postulated a set of physico-mathematically based axioms. These axioms concern the speed, packing density, and noise threshold of the energy with which any computing device detects or alters the physical representation of information. The principal result of our work to date is that the time needed by a spatially distributed system to compute any n-argument function grows with n at least as $n^{1/3}$. This result is based only on the above-mentioned axioms and on the fact that the computing function depends non-trivially on all its arguments. Further results indicate that, regardless of the way in which identical computing modules are "stacked" in space, they cannot compute a function of n arguments as fast as the above bound -- in fact, they often compute such a function no faster than $n^{1/2}$. Finally, the above bound has been combined with certain other results, yielding a measure for the computational efficiency of a process distributed in time and space. Through this measure, it is possible to assess the efficiency of a given space-time process. The details of this development will appear in the Proceedings of the 1971 Switching and Automata Theory Conference in a paper by Dertouzos.

Another area of fundamental work is motivated by the logical capabilities and limitations of the analog substructure of an exact-inexact machine. Here, we have investigated the logical capabilities of certain dynamic analog structures made up of sample-holds and integrators. This work has resulted in a wealth of interesting results, theorems, and constructive techniques for dynamic-system synthesis. They will appear in the doctoral dissertation of M. E. Kaliski, M.I.T. Department of Electrical Engineering, to be completed shortly.

We have also investigated the logical capabilities of a class of Turing Machines which can store and process real numbers. Results of this work are related to computations on the real numbers. They will appear in some detail in the Proceedings of the 1971 Switching and Automata Theory Conference in a paper by Abramson.

Publications 1970-1971

Abramson, F. G., Models for Continuous-Discrete Computation, S.M. Thesis, Dept. of Electrical Engineering, February 1971.

Dertouzos, M. L., "Computer Graphics: Problems and Progress", Proceedings, Erlangen Symposium on Display Use for Man-Machine Dialog, Institut fur Mathematische Mashinen und Datenverarbeitung, Erlangen, Germany, March 1971.

Dertouzos, M. L., "Elements, Systems and Computation: A Five Year Experiment in Combining Networks, Digital Systems and Numerical Techniques in the First Course", Proceedings, Purdue

Publications 1970-1971 (cont.)

1971 Symposium on Applications of Computers to Electrical Engineering Education, Purdue University, Indiana, April 26-28, 1971.

Dickens, M. W., Computer Graphics: Central Problems and Their Treatment, S.M. Thesis, Dept. of Electrical Engineering, June 1971.

Lum, M., Computer-Aided Analysis of Nonlinear Networks, S.M. Thesis, Dept. of Electrical Engineering, January 1971.

Lynn, C. W., Non-Linear Function Processing for Computer Analysis of Networks, S.M. Thesis, Dept. of Electrical Engineering, June 1971.

Weinberg, A., Computer-Aided Education in Subject 6.001, S.M. Thesis, Dept. of Electrical Engineering, January 1971.

INTERACTIVE MANAGEMENT SYSTEMS

ORGANIZATIONAL INFORMATION SYSTEMS

Prof. M. M. Jones

Instructors, Research Associates, Research Assistants and Others

D. Asthana
G. T. Dixon
R. C. Goldstein
S. P. Mason
A. R. Monroe-Davies
R. C. Owens
D. H. Porges

Undergraduate Students

R. L. Brooks
R. M. Elkin
R. S. Goldhor
P. H. Guldberg
C. A. Hatvany
D. M. Krackhardt
W. Y. Ng
S. Pincus
J. L. Rosenberg
H. J. Siegel

DSR Staff

J. A. Friel
M. Lenot
S. E. Niles
A. J. Strnad
D. M. Wells

Support Staff

E. T. Moore
R. Queens

Guest

Prof. J. I. Elkind

* * * * * * * *

SIMPL PROJECT

Prof. M. M. Jones

Instructors, Research Associates, Research Assistants and Others

R. Bryant
A. Gonzales
S. Murthy

Undergraduate Students

R. M. Berman
D. J. Chang
S. S. Cohen
J. E. Jagodnik
S. M. Stoney

DSR Staff

R. C. Thurber, Jr.

X. INTERACTIVE MANAGEMENT SYSTEMS

ORGANIZATIONAL INFORMATION SYSTEMS

A. Introduction

The Organizational Information Systems Group seeks to develop and understand how to use interactive information systems in the administration and operation of organizations. The systems that we are investigating incorporate a data base that describes the present and past state of the organization, models that can be used to predict future states, and procedures that assist in making planning and control decisions. Also central to these systems are facilities that allow users to interact with this body of data, models and procedures. To lend concreteness to our systems research, much of the work has been done in the context of specific applications -- largely the administrative problems of Project MAC itself and of several of the academic departments at M.I.T. During the coming year, we expect to broaden considerably the set of applications with which we shall deal.

Work on organizational information systems began in 1968 with the start of the MacAIMS Project. In the beginning, that project attempted to integrate a number of interactive systems already operating on CTSS into a management system and to develop management information systems for Project MAC on CTSS. Some operationally useful interactive systems for personnel management, budgeting, inventory control and purchasing were derived from this work. Since June 1970, our research has focused on the development of general-purpose data-manipulation facilities on Multics and on the application of these facilities in management systems for Project MAC. During the last few months, we have begun work on techniques for modeling organizations, and have started developing models for Project MAC. This work has been supported in part by ARPA through ONR and in part by ONR directly.

The principal projects undertaken during the last year were:

1) Design and implementation of a set-theoretic data-manipulation system on Multics.

2) Development of management information systems for Project MAC and for the Sloan School.

3) Studies of access control and privacy in computer data-base systems.

4) Studies of models and modeling of organizations.

These research projects are discussed more fully below.

B. Set-Theoretic Data-Manipulation System

During the past year, R. C. Goldstein, A. J. Strnad, D. M. Wells and S. E. Niles, with the aid of a number of students, have

designed and implemented a data-manipulation system that is based on a set-theoretic organization of data and operations. The initial version of this system is now operating and is being used for a Personnel Data System for Project MAC. Studies of the performance of this initial system will be used as a basis for additions, modifications and improvements that will be made during the next year. The system is programmed in PL/1 and is implemented on Multics.

In this system, we have taken the view that information stored in a data base consists of sets of Data Elements (DE) and sets of relations among them that are called Relational Data Sets (RDS). The basic set-theoretic primitive operations are used for manipulating the RDS.

Given the Data Element Sets (DES) S1, S2, ... Sn, the Relational Data Sets consist of n-tuples (tuples of degree n), each of which has its first element from set S1, its second element from S2, and so on. The Relation Descriptor (RD) is, in our terminology, the n-tuple composed of the names of the sets S1, S2, ... Sn. Suppose, for example, that there are Data Element Sets for persons' names, for addresses and for telephone numbers. We might construct an RDS which will represent the relations among members of these sets. The Relation Descriptor for this RDS will be the 3-tuple <person-name, address, telephone-number>. All other tuples will express the relation among the members of these sets. In our implementation, the relations are stored exclusively in terms of Reference Numbers (RN).

The whole system is logically divided into two major parts. In the first, Data Element Sets are stored, Reference Numbers are assigned to the Data Elements, and operations are performed on the DES. In the second part of the system, Relation Data Sets are created and stored, and basic set-theoretic primitive operations are performed on them.

Reference Numbers (identification numbers) play an important role in our implementation. Whenever a new DE enters the system, it is immediately assigned a Reference Number. The RN is used for all subsequent operations on that DE. The method used for storing and assigning RN to DE preserves the ordering of the DE and guarantees that a particular DE is stored only one time within the system.

C. Management Information Systems

The development of two management information systems has been undertaken this year. One of these is to aid in the administration of Project MAC. The other is for the administration of the Sloan School.

The Project MAC system, designed and implemented by A. J. Strnad and S. E. Niles, uses the data-manipulation system described above. In its initial version, the system will provide interactive data storage, retrieval, manipulation, and report-generation support for the personnel-management, budgeting, inventory-control, and facilities-management functions of the business

office. The personnel function has been implemented and experiments are under way to evaluate its performance.

The Sloan School system, under the direction of Prof. M. S. Scott-Morton and Prof. J. F. Rockart, has focused on the analysis and design of a decision support system for budgeting. The progress to date has been mainly in the initial decision analysis and tool building. A model of the current budgetary methods in the Sloan School has been developed. The budgetary decisions made by administrative personnel have been identified, and the information required for these decisions has been determined.

D. Studies of Access Control and Privacy

A Master's thesis by R. C. Owens, Jr., "Primary Access Control in Large-Scale Time-Shared Decision Systems", was completed in May 1971. The thesis identified four primary dimensions of the access control: 1) the physical level at which to apply control, 2) the fineness of distinction to the term "access", 3) the meaning of the term "user identification", and 4) the degree of sophistication employed in automatically assigning restrictions to new data files.

Within the context of MacAIMS, the Project MAC Advanced Interactive Management System, the design of an access-control system is presented which takes positions along these four dimensions appropriate for controlling access in a Management Decision System. Support is provided for constraints specified as general logical restrictions based on 1) the characteristics of the entity requesting access, 2) the content of the sensitive data item, 3) the context in which the sensitive item appears, 4) proper completion of an interactive procedure, and 5) combinations of any of these. The access levels that may be specified are based on the logical (not the physical) nature of the interaction that the user requests.

The system presented here is an interim system in that it does not solve all the access-control problems of MacAIMS. Among the unsolved problems is that of Truth: in a data management system that provides a powerful set of operators, it is easy to create false information in very subtle ways. Another problem is that of conflicts of privacy. Solutions to these problems must be found before the access-control scheme will be complete.

R. C. Goldstein has begun a doctoral thesis, "The Political Dynamics of Information and Privacy", in which he intends: 1) to investigate the interaction between individual privacy and "quality of life" in a society, and 2) to explore techniques that can be used to protect privacy.

E. Modeling of Organizations

P. Kleindorfer, M. Lenot, H. J. Siegel and Prof. J. I. Elkind have just begun a study of organizational models. The operations of Project MAC, as an example of a research and development organization, are being analyzed. We are obtaining a description of the principal activities of the Project and we shall

attempt to express this description in the form of a quantitative model.

Publications 1970-1971

Goldstein, Robert C., "Helping People Think", Naval Research Reviews, January 1971; also Project MAC Technical Memorandum 25, April 1971, AD 721-998.

Goldstein, Robert C., "The Substantive Use of Computers for Intellectual Activities", Project MAC Technical Memorandum 21, April 1971, AD 721-618.

Goldstein, Robert C., and Strnad, Alois J., "The MacAIMS Data Management System", presented at the ACM SICFIDET Workshop on Data Description and Access, Houston, Texas, November 1970; also Project MAC Technical Memorandum 24, April 1971, AD 721-620.

Goldstein, Robert C., "Position Paper on Computers, Data Banks and Bill of Rights", prepared for Subcommittee on Constitutional Rights, Committee on the Judiciary, U. S. Senate, March 1971; AD 721-670.

Owens, Richard C., Jr., "Primary Access Control in Large-Scale Time-Shared Decision Systems", thesis, Master of Science, Sloan School, M.I.T., June 1971; also MAC TR-89, AD 728-036.

Strnad, Alois J., "The Relational Approach to the Management of Data Bases", Project MAC Technical Memorandum 23, April 1971, AD 721-619; material also accepted for presentation at IFIPS, August 1971.

Wells, Douglas M., "Transmission of Information between a Man-Machine Decision System and its Environment", Project MAC Technical Memorandum 22, April 1971, AD 722-837; material also accepted for presentation at IFIPS, August 1971.

SIMPL PROJECT*

A crude version of SIMPL was initially operable by the beginning of August 1970, but it had not been thoroughly tested and most of the advanced features of SIMPL were not available. In August, the system was used by approximately 20 members of a special Sloan School Summer Session Simulation Seminar, during which period we discovered many of the bugs and limitations of that first system. A users' manual was also hurriedly prepared for the Sloan School.

The next few months were spent debugging and modifying that prototype system, as well as revising and expanding the descriptive documentation. The system was then used extensively by the Sloan School simulation class (15.572) during the Fall semester. This continuous class use quickly revealed the bugs remaining in the system. By the end of the semester, the SIMPL translator and run-time support system was relatively stable and bug-free. Queuing statistics and a limted form of tracing were added to the system, and numerous minor improvements were made.

By January 1971, the enhanced version of the original system was functioning reliably, but it exhibited several major weaknesses:

1) Translation and compilation times were excessively slow. The translator itself was slow, and it produced a PL/1 program that was very large and hence took a long time to compile.

2) External activities (similar to external subroutines) were not implemented, and were very difficult to implement under that system. The user was thus forced to use large amounts of computer time to retranslate and recompile his entire model whenever he wanted to make a small change in any part of the model.

3) The advanced interactive features were not implemented, and the system design made them difficult to implement. Even more important, the implementation of those features would have significantly worsened the already-slow translation and compilation times.

Thus, the system has not yet realized our goal of giving the user an "incremental" simulation system. We therefore decided to undertake the design and implementation of a new system which would be more efficient in all respects, and which would easily accommodate all the additional features of the full SIMPL system. We felt that our experience with the old system would enable us to produce the improved system in a relatively short amount of time.

---

*In August 1970, the SIMPLE Group decided to change its name to the SIMPL Group.

The new system (now called Version 2) was designed and programmed beginning in February 1971, and is now in the final stages of testing and debugging. Besides being more efficient in translation, compilation and execution, Version 2 includes a complete tracing capability, allows external activities, produces numerous simulation oriented statistics, and supports the interactive SIMPL Monitor. None of these features was available in the earlier Version 1. The SIMPL Monitor itself is written and working; it is a very flexible run-time system which allows the user almost complete freedom to inspect and modify his model, then to continue or restart the simulation.

The SIMPL system has also been conscientiously documented. At present, documentation comprises three manuals. The SIMPL Primer is a short description of SIMPL, intended to give new users a quick introduction to the system. The SIMPL Reference Manual contains a complete description of all features of the system and their use. The SIMPL Implementation Manual describes the Multics implementation of SIMPL. (The latter currently describes only Version 1; several new chapters have yet to be added to bring it up to date.)

Current plans call for finishing work on the SIMPL system by 30 September 1971, and releasing it to the M.I.T. community for general use. The system will undergo a final test during the 1971 Sloan School Summer Session Simulation Seminar in late August. Between now and then, we plan to implement a few new features (including process priorities, ranked sets, and interpolation functions), to complete the implementation manual, and generally to streamline the system for release to the public.

There will, no doubt, be a continuing need for maintenance throughout the 1971-1972 school year and plans are being made to provide that assistance. Also, the installation of the Version 2 PL/1 compiler may necessitate some slight reprogramming.

There are no present plans to add a well-integrated graphical facility to SIMPL, using something like the IMLAC PDS-1, although that would make an exciting thesis project. Also, we have completely written off the idea of ever implementing a true interpreter for the SIMPL language, deeming that far too big a job and not worth the effort now that we allow external activities to be separately compiled and debugged. It is our hope that we can redirect our efforts from developers of SIMPL to users.

MATHLAB

Prof. W. A. Martin
Prof. J. Moses

Instructors, Research Associates, Research Assistants and Others

R. J. Fateman
P. S. Wang

Undergraduate Students

M. R. Genesereth
E. Kohn
E. C. Rosen
S. E. Saunders
E. Tsiang
L. E. Widman
R. E. Zippel

DSR Staff

M. J. Ablowitz
H. O. Capps
J. P. Golden
L. P. Rothschild
R. C. Schroeppel

Support Staff

K. Young

XI. MATHLAB

During the past year the Mathlab group has continued to develop the MACSYMA system for interactive algebraic manipulation. The principal modules of MACSYMA are shown in Fig. 1. Those indicated by circles are complete.

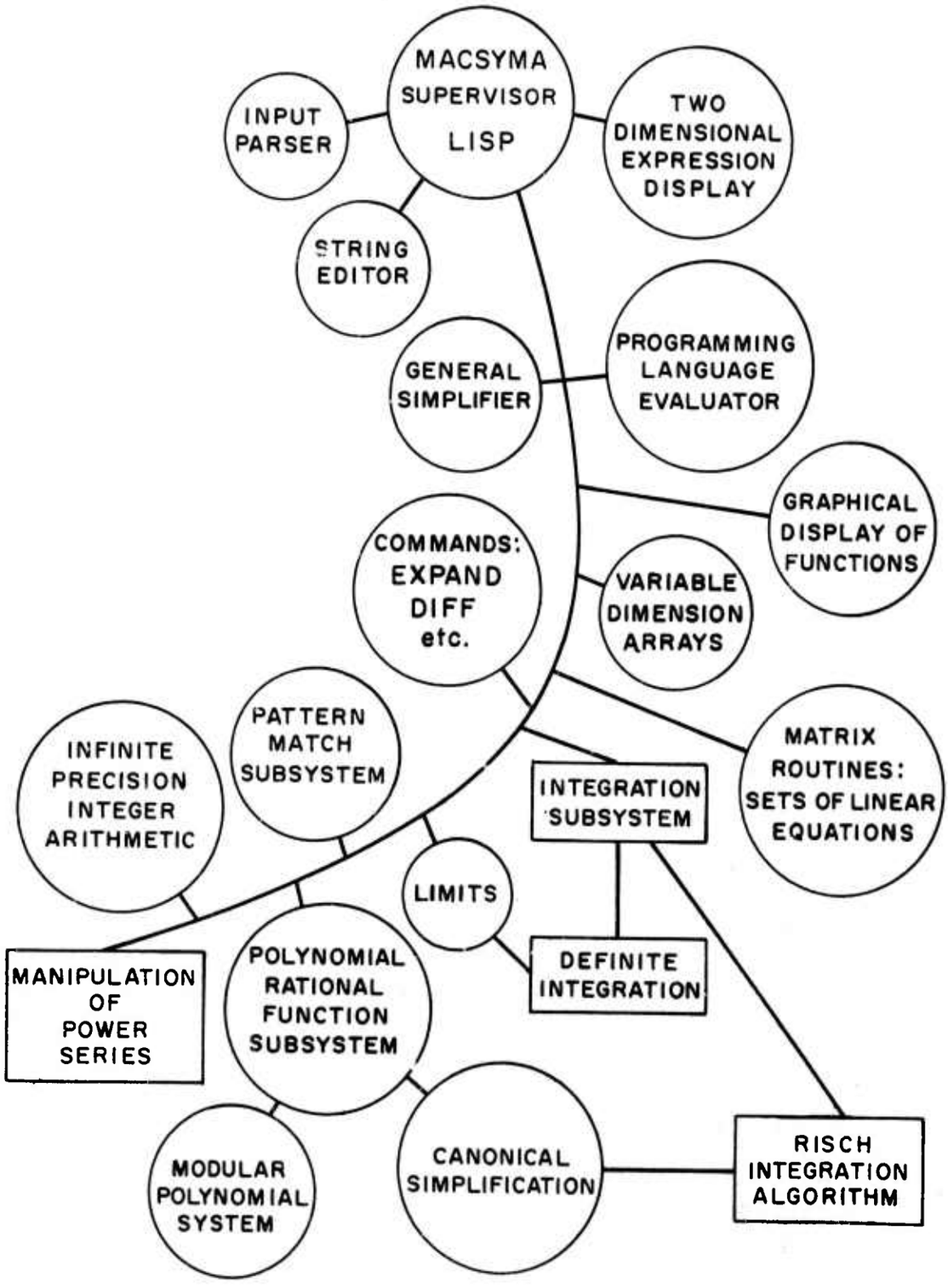


FIG. 1. THE MACSYMA SYSTEM.

MATHLAB

Seven papers describing MACSYMA and related work of our group were presented at the Second Symposium on Symbolic and Algebraic Manipulation held in Los Angeles, 23-25 March 1971. MACSYMA has now reached a point where it is both a useful tool for the solution of real problems and a convenient base for research in algorithm analysis, and development of advanced systems for applied symbolic mathematics. We are beginning to use MACSYMA for the solution of several problems of interest in mathematics and physics.

With Prof. Bers, of the M.I.T. Department of Electrical Engineering, we are using MACSYMA to investigate the properties of the dispersion relation of a linear system.

With Dr. Eytan Barouch, of the M.I.T. Department of Mathematics, we explored some problems in statistical mechanics.

Quantum mechanical calculations for Mr. F. Heile's S. M. thesis (M.I.T., Physics) were done in MACSYMA.

Within our own group, L. Rothschild and Prof. J. Moses have used MACSYMA for testing mathematical conjectures, and R. J. Fateman has used the system for solving a large number of sets of simultaneous linear equations arising in the analysis of MACSYMA's polynomial manipulation routines.

New algebraic manipulation algorithms are also under investigation. Recently developed polynomial manipulation algorithms using modular arithmetic have been implemented. General methods of obtaining simplification rules for functions, defined by differential equations, are also being developed.

Publications 1970-1971

Martin, William A. and Richard J. Fateman, "The MACSYMA System", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 59-75.

Martin, William A., "Computer Input/Output of Mathematical Expressions", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 78-89.

Moses, Joel, "Algebraic Simplification: A Guide for the Perplexed", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 282-304.

Fateman, Richard J., "The User-Level Semantic Matching Capability in MACSYMA", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 311-323.

Publications 1970-1971 (cont.)

Martin, William A., "Determining the Equivalence of Algebraic Expressions by Hash Coding", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 305-310.

Moses, Joel, "Symbolic Integration: The Stormy Decade", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 427-440.

Wang, Paul S., "Automatic Computation of Limits", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 458-464.

PROGRAMMING LANGUAGES

Prof. J. J. Donovan

Instructors, Research Associates, Research Assistants and Others

V. Altman
J. D. DeTreville
R. Earle
L. I. Goodman
G. Holt
J. Johnson
W. J. Klos
D. König
S. E. Madnick
W. C. Michels
P. Olson
H. M. Toong
L. E. Travis

Undergraduate Students

P. G. Bras
R. Davis
C. A.Kessel
N. V. Kohn
J. C. Lind
J. Quimby
J. L. Reuss
A. M. Solish

Support Staff

D. Goldthorpe
E. F. Nangle

Guest

H. Adler

XII. PROGRAMMING LANGUAGES

A. Introduction

During 1970-1971, research in the Programming Languages Group focused on analysis of languages and their translators (compilers) and the environment in which they exist (operating systems).

B. Canonic Systems

A canonic system is a type of formal system that operates on several sets of strings over a finite alphabet. Canonic systems, (equivalent to Smullyan's elementary formal systems) are a variant of Post's canonical systems. In canonic systems, the general framework of productions or string-transformation rules is replaced by a system of axioms (canons) and by the logical rules of substitution for variables and detachment (modus ponens). A canonic system defines a set of inter-related predicates, each of which is a set of strings.

In particular:

A canonic system is a sextuple

$\mathcal{C} = (C , V , M , P , S , D)$

C is a finite set of canons

V is an alphabet of terminal symbols used to form the strings generated (i.e., provable) by $\mathcal{C}$

M is a finite set of variable symbols (variables)

P is a finite set of predicate symbols (predicates) used to name sets of n-tuples. The number of components in the n-tuples denoted by a predicate is the degree of the predicate

S is a finite set of punctuation signs used in writing canons

D ($\subseteq$P) is a set of sentence predicates, the union of which will be defined to be the language specified by the canonic system

Canonic systems have been used to specify the syntax and the translation of programming languages. They have served as a data base for a generalized translator for computer languages, for proving various theorems as to their mathematical power and their formal properties, and they have been used to study the complexity of translators and languages.

C. Power of Canonic Systems

We have proven a general theorem relating canonic systems to various types of formal grammars.

Theorem. For every type of grammar, there exists a class of canonic systems with the property; that for every grammar of the type under consideration there exists a canonic system that generates the same language and that belongs to a corresponding class. Further, that class of canonic systems can be constructed.

Many formal systems -- for example, canonic systems and Type 0 grammars -- have inherent undecidability problems: in general, there is no algorithm capable of telling, after a finite amount of time, whether or not a given string is in the language of these grammars. Studies of power help us to understand how characteristics of a grammar correspond to structural features of languages and to choose the weakest grammar suitable to a given situation. At the same time, by exploring restrictions we learn about the structure of language.

Figure 1 is an inclusion diagram of the relationships between certain classes of grammars. Classes of systems in the diagram include all classes below them (that is, at nodes below them in the tree). The diverging branches represent classes for which inclusion either does not exist or is not presently known.

D. Canonic Systems and Recursive Sets

We have proven that there can exist no class of canonic systems defining (as a class) only recursive sets, all the recursive sets, and including all the canonic systems which define recursive sets. Likewise, it is recursively undecidable whether a language of Type 0 is also of Type 1 (2,3); but it is decidable whether a grammar of Type 0 is also of Type 1 (2,3). Grammars of Type i define all and only languages of Type i, but it is not the case that (for i > 0) only grammars of Type i define languages of Type i.

A priori, it might be the case that a certain class of canonic systems, the NCS2 for instance, would correspond to recursive sets in the sense that it defines all and only recursive sets without claiming monopoly in defining recursive sets (i.e., not all canonic systems defining recursive sets are in that class).

The main result is that there can be no such class. In particular, NCS2 $\subseteq$ Rec. The proof is by diagonalization (after a suitable "Gödelization" of canonic systems).

E. Generalized Translator

An efficient algorithm which is capable of recognizing strings produced by a canonic system has been developed as an extension of an earlier algorithm presented by Cheatham and Sattley. The algorithm is principally top-down; it attempts first to match the input string against the sentence predicate of the canonic system (e.g., program), and it arrives only through recursion at a lower-level predicate (e.g., integer or digit).

This algorithm removes the inadequacies of Backus Naur Form (BNF) specification of the syntax of programming languages. In BNF it is impossible to describe many of the constraints that

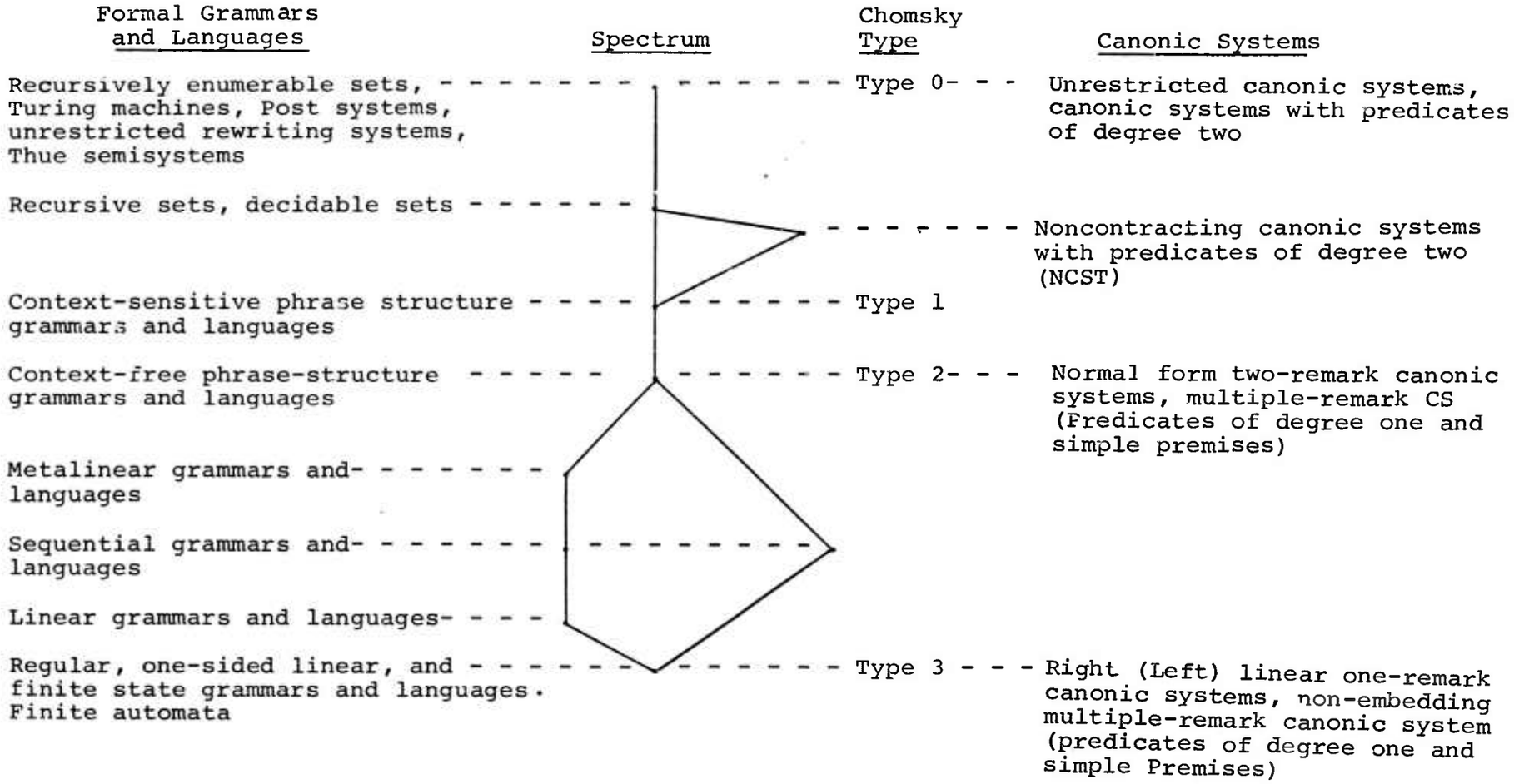


Fig. 1. CORRESPONDING HIERARCHIES

exist in programming languages, such as the restriction that a "legal program" is not acceptable to a translator, even though correct in form, if not all of the reference labels in the program correspond to statement labels (sometimes referred to as "context-sensitive features").

F. Canonic Reduction Generator

The production language, as introduced by Floyd, affords the capability of implementing a one-pass, one-push-down-stack recognizer for a computer language. The power of the production language may be even further enhanced, however, by the introduction of action routines, to be called for the purpose of code generation upon the detection of a legal and complete syntactic form. These enhanced productions are referred to as reductions. The problem exists, however, of getting from a specification of a language to the productions or reductions. The problem of generating productions has been solved by Earley, but his algorithm specifies the generation of productions given a BNF representation of a language. However, BNF is incapable of representing the translation of a computer language, and thus reductions cannot be generated from a BNF specification.

Canonic systems, on the other hand, can be used to specify the translation of a computer language as well as its syntax. We have developed an algorithm of generating reductions given a canonic systems specification of a language. The algorithm draws on the work of Earley, and in fact, is identical to Earley's algorithm for the case of single level canonic systems (except, of course, that canonic systems rather than BNF form the language specification). The algorithm has been implemented to handle predicates of level one or two. In the case of a level two predicate, the second element specifies the action routine associated with the given syntactic form.

G. Undecidability of Programming Languages

It is well known that in a language where conditional transfers of control are available, it is decidable that a program contains a loop, but it is undecidable whether or not any particular loop will ever be entered, or "more generally", whether or not the program will ever wind up in a loop. Given an arbitrary Turing machine, this follows the impossibility of deciding whether or not a particular instruction will ever be executed. This latter problem is undecidable since we can replace all HALT instructions by just one, and if we could decide whether or not that instruction would ever be entered, we would have a solution to the halting problem.

PL/1 has a compile-time facility which enables the programmer:

> to direct the compiler to compile a certain group of source-language statements rather than some other group;
>
> to include source-language statements or data stored on some storage device;

and so on.

A preprocessor performs these compile-time operations and gives as output, a stream of source-language statements from which it has determined that these and only these statements are to be included and compiled in the program. In order to be able to perform its function, the preprocessor recognizes and executes, among other compile-time statements, conditional transfers.

The stream of statements given as input to a PL/1 compiler contains statements to be compiled and statements addressed to the preprocessor to be executed at compile time. Confining our attention to these latter statements only, we see that they satisfy the conditions that they constitute an [alleged] program, written in a language which includes conditional transfers of control. It is decidable whether there are any compile-time loops, but it is undecidable that the program will ever enter a loop.

The compiler is presented with a stream of statements and is expected to compile it, if it is syntactically correct, or to reject it if it is not. Suppose, however, that the stream includes compile-time statements, and that these compile-time statements include loops; if such a loop will ever be entered, the compiler will not halt (assuming an infinite scratch file), and the program will not be compiled (will not be accepted). Since it is undecidable whether or not such a loop will ever be entered, it is undecidable that the input stream of statements is accepted as a program. In other words, the set of PL/1 programs, if we consider the compile-time facility as an integral part of the language (as customary), is not recursive.

H. Measure Function of Programming Languages Resource Usage

A theory of complexity has been developed for algorithms implemented in typical programming languages. The complexity of a program may be interpreted in many different ways; a method for measuring a specific type of complexity is a complexity measure -- some function of the amount of a particular resource used by a program in processing an input. Typical resources would be execution time, core, I/O devices, and channels.

An approach has been developed that analyzes the complexity of a program with respect to a valid set of inputs -- a finite set of legitimate, halting inputs. A program equation is developed to make the transformations undergone by the inputs more explicit. Using the equation, the input set is partitioned into classes of constant complexity. The classes are used to compute maximum, minimum, and expected complexities of the program on the input set.

Several equivalence relations have been defined, relating different programs by their complexity. Complexity has also been treated in terms of concatenation and functional equivalence of programs.

I. Programming Systems Environment

As the interaction between programming languages and the

operating system in which the language finds itself becomes less distinct, we find ourselves studying operating systems in our group. The most important aspect of the operating system is that the programming language must interact with the file system.

We have developed an approach to the design and study of file systems that allows the designer of a file system to systematically implement and analyze the file system. This approach has been used both for teaching file systems and for the design of file systems.

These ideas have beeen further developed by investigating the relationship of programming language requirements in the environment of a real-time computer-based sensor system. To this end, we have developed an advanced and comprehensive processing system for the IBM 1130.

This included a software-assisted multilevel-priority interrupt mechanism, an on-line simulation language, optimizing compiler, advanced binder, and generalized file system.

J. Community Activities

Members of the group were involved in two major community activities. We feel that M.I.T.'s greatness and more generally the responsibility of scientists throughout our country, will lie not only in the advancement of technical knowledge of achievements, but also in the dissemination of this knowledge to the communities and to the people that may use this knowledge effectively. We engaged in two projects, each addressing itself to the dissemination of knowledge to different groups of people. The first group was the community and its individuals. This project was undertaken during the summer of 1971. We selected a community that is facing many technical problems, e.g., communications, sewerage disposal, power distribution, and mosquito control. We sought support from the town's businessmen to support students from the town to investigate the problem. These students worked in conjunction with M.I.T. students.

The other group, to which we have addressed ourselves, is the undereducated, underprivileged people in the Boston community. We have addressed ourselves to this group through the Lowell School, which is a school under the auspices of M.I.T. in the evenings. We have helped to restructure the school to admit and teach these people and try to expose them to the process of learning and the rewards and satisfaction thereof. We have centered the program around computers, using them as a tool for accomplishing our objectives.

K. Teaching

Member. of our group have been involved in conceiving and teaching several courses whose activities are directly related to the major research activity of this group, namely, programming languages.

Course 1 - Programming Languages in Formal Systems - A Graduate Credit Course

Course 2 - Operating Systems - Independent Activities Period Seminar

Course 3 - Digital Computer Programming Systems - Undergraduate

Publications 1970-1971

Altman, Vernon, "A Canonic Reduction Generator", S. B. Thesis, Department of Electrical Engineering, August 1970.

Donovan, John J., Preliminary Edition of Systems Programming, McGraw-Hill, 1971, New York.

Earle, Roy, "Global Optimization in Algebraic Language Compilers", S.M. Thesis, Department of Electrical Engineering, May 1971.

Holt, George, "Semantic Models for Data Description", S.M. Thesis, Alfred P. Sloan School of Management, June 1971.

Johnson, Jerry, "File System to Support Time Sharing in a Multi-Programming Environment", S. M. Thesis, Department of Electrical Engineering, June 1970.

Klos, Walter J., "A Program Capable of Printing Aggregate Data", S. B. Thesis, Department of Electrical Engineering, October 1970.

Madnick, Stuart E., "Development of Computer-Based Sensor Systems", Proceedings of Third Hawaii International Conference on System Sciences, University of Honolulu, Honolulu, Hawaii, January 1970.

Madnick, Stuart E., "Design and Construction of a Pedagogical Micro-Programmable Computer", Proceedings of Third Annual Workshop on Microprogramming, University of Buffalo, New York, October 1970.

Madnick, Stuart E., "Program Parallelism Based upon Computation Schemata", Proceedings of the VI International Congress on Cybernetics, Namur, Belgium, September 1970.

Ramchandani, Chander, "Debugging Scheme to Run Interpretively in Virtual Memory", S. M. Thesis, Department of Electrical Engineering, January 1970.

Travis, Leon E., "A Cobol Interpretive System", S. M. Thesis, Department of Electrical Engineering, August 1970.

Zilles, Stephen, "Synchronization of Resource Usage in a Small Information System", Proceedings of Third Hawaii International Conference on System Sciences, University of Honolulu, Honolulu, Hawaii, January 1970.

ARTIFICIAL INTELLIGENCE

July 1969 to December 1970

Prof. M. Minsky
Prof. S. Papert

Academic Staff

Prof. M. J. Fischer
Prof. P. E. O'Neil
Prof. S. Papert
Prof. M. S. Paterson
Prof. P. Winston

Instructors, Research Associates, Research Assistants and Others

H. Abelson
E. Charniak
R. J. Donaghey
M. Dowson
E. Freuder
I. Goldstein
B. K. P. Horn
L. Krakauer
R. LeCompte
J. Lerman
S. Lothes
R. B. Roberts
M. Speciner
D. Spencer
G. J. Sussman
D. Waltz
T. A. Winograd

Undergraduate Students

D. T. Dalton
W. Freeman
J. Freiberg
P. Gagner
J. Gaschnig
S. Glazer
N. Goodman
D. C. Holzer
P. Jensen
R. F. Mohl
J. Rubin
J. C. Shockey
M. N. Slusarczuk
B. M. Trager
E. D. Trautman
N. S. Weinstein
J. Whitbeck
L. F. Yeager

DSR Staff

M. Beeler
R. H. Freyberg
R. W. Gosper, Jr.
R. Greenblatt
A. K. Griffith
W. Henneman
T. F. Knight
W. Neely
R. Noftsker
J. S. Roe
L. P. Rothschild
P. R. Samson
R. Schroeppel
J. Shah
S. W. Smoliar
C. Solomon
G. L. Wallace
J. L. White
R. W. Williams
L. R. Wilson

ARTIFICIAL INTELLIGENCE

Support Staff

T. F. Callahan
T. Carlton
P. DeCoriolis
F. J. Drenckhahn
D. E. Eastlake
J. L. Fowler
M. Harpole
P. Holloway
E. I. Kampits
R. J. Lebel
G. H. H. Mitchell
G. Roe
L. A. Sands
D. Silver
N. F. Stone
C. T. Waldrop
J. B. Weiss

Guests

R. April
Prof. W. W. Bledsoe
R. Boyer
J. Cohen
J. Jaroslav

I. ARTIFICIAL INTELLIGENCE

The A. I. Laboratory is concerned with understanding the principles of Intelligence. Its goal is to develop a systematic approach to the areas that could be called Artificial Intelligence, Natural Intelligence, and Theory of Computation. Here are its main current foci of attention:

ARTIFICIAL INTELLIGENCE

Robotics; Vision, mechanical manipulation, advanced automation. Models for learning, induction, analogy. Schemata for organizing bodies of knowledge. Development of "heterarchical" program control structures.

NATURAL INTELLIGENCE

Models of structures involved in "common sense thinking". Understanding meanings, especially in natural language narrative. A new educational methodology, based on development of the child's abilities to describe processes.

THEORY

Computational trade-offs between time, memory size, and processor parallelism. Study of computational geometry as a tool for comparing different structures and strategies. Theory of schemata, for analysis of complexities of certain algorithms and languages.

These subjects are all closely related. The natural language project is intertwined with the common sense meaning and reasoning study, in turn essential to the other areas, including machine vision. Our main experimental subject worlds, the "blocks world" robotics environment and the children's story environment, are better suited to these studies than are the puzzle, game, and theorem-proving environments that became traditional in the early years of artificial intelligence research. Our evolution of theories of intelligence has become closely bound to the study of development of intelligence in children. The educational methodology project is symbiotic with the other studies, both in refining older theories and in stimulating new ones; we hope this project will develop into a center like that of Piaget in Geneva.

As it has crystallized over the past few years, the main elements of our viewpoint can be summarized cryptically:

Thinking is based on the use of SYMBOLIC DESCRIPTIONS and description-manipulating processes to represent a variety of kinds of KNOWLEDGE -- about facts, about processes, about problem-solving, and about computation itself, in ways that are subject to HETERARCHICAL CONTROL STRUCTURES -- systems in which control of the problem-solving programs is affected by heuristics that depend on the meanings of events.

The ability to solve new problems ultimately requires the intelligent agent to conceive, debug, and execute new procedures. Such an agent must know to a greater or lesser extent how to plan, produce, test, modify, and adapt procedures; in short, it must know a lot about computational processes. We are not saying that an intelligent machine, or person, must have such knowledge available at the level of overt statements or consciousness, but we maintain that the equivalent of such knowledge must be represented in an effective way somewhere in the system.

This report illustrates how these ideas can be embodied into effective approaches to many problems, into shaping new tools for research, and into new theories we believe important for Computer Science in general, as well as for Robotics, Semantics, and Education.

Much of the material in this report is also part of a draft of a book on Thinking. For information about subsequent drafts and publication write to the authors at the A. I. Laboratory.

The Laboratory is seeking young workers who believe they can do work of the quality described herein, as staff, graduate students, or post-doctoral fellows.

1.0 Vision and Description

When we enter a room, we feel we see the entire scene. Actually, at each moment most of it is out of focus, and doubly imaged; our peripheral vision is weak in detail and color; one sees nothing in his blind spot; and there are many things in the scene we have not understood. It takes a long time to find all the hidden animals in a child's puzzle picture, yet one feels from the first moment that he sees everything. People can tell us very little about how the visual system works, or what is really "seen". One explanation might be that visual processes are so fast, automatic, and efficient that there is no place for introspective methods to operate effectively. We think the problem is deeper. In general, and not just in regard to vision, people are not good at describing mental processes; even when their descriptions seem eloquent, they rarely agree either with one another or with objective performances. The ability to analyse one's own mental processes, evidently, does not arise spontaneously or reliably; instead, suitable concepts for this must be developed or learned, through processes similar to development of scientific theories.

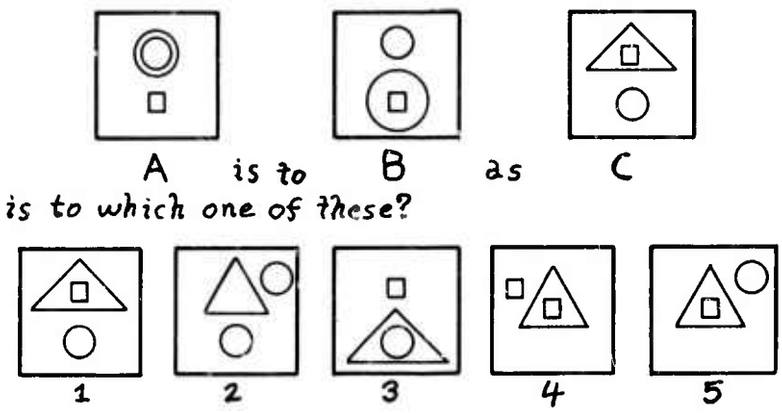
Most of this report presents ideas about the use of descriptions in mental processes. These ideas suggest new ways to think about thinking in general, and about imagery and vision in particular. Furthermore, these ideas pass a fundamental test that rejects many traditional notions in psychology and philosophy; if a theory of Vision is to be taken seriously, one should be able to use it to make a Seeing Machine!

1.1 Reasoning by Analogy

To emphasize that we really mean "seeing" in the normal human

sense, we shall begin by showing how a computer program -- or a person -- might go about solving a problem of "reasoning by analogy". This might seem far removed from questions about ordinary "sensory perception". But as our thesis develops, it will become clear that there is little merit in trying to distinguish "sensation" or "perception" as separate and different from other aspects of thought and knowledge.

When we give an "educated person this kind of problem from an IQ test, he usually chooses the answer "Figure 3":



People do not usually consider such puzzles to be problems about "vision". But neither do they regard them as simply matters of "logic". They feel that other, very different mental activities must be involved. Many people find it hard to imagine how a computer program could solve this sort of problem. Such reservations stem from feelings we all share; that choosing an answer to such a question must come from an intuitive comprehension of shapes and geometric relations, rather than from the mechanical use of some rigid, formal rules.

However, there is a way to convert the analogy problem to a much less mysterious kind of problem. To find the secret, one has merely to ask any child to justify his choice of Figure 3. The answer will usually be something like this!

"You go from A to B by moving the big circle down.
You go from C to 3 in the same way by moving the big triangle."

On the surface this says little more than that something common was found in some transformations relating A with B AND C with 3. As a basis for a theory of the child's behavior it has at least three deficiencies:

It does not say how the common structure was discovered.

It appears to beg the question by relying on the listener to understand that the two sentences describe rules that are

identical in essence although they differ in details.

It passes in silence over the possibility of many other such statements (some choosing different proposed answers). For example, the child might just as well have said:

"You go from A TO B by putting the circle around the square..."

or

"You go from A TO B by moving the big figure down," etc.

Aha! If that last statement were applied also to C and 3, the rules would in fact be identical! This leads us to suggest a procedure for a computer and also a "mini-theory" for the child:

Step 1. Make up a description DA for Figure A and a description DC for C.

Step 2. Change DA so that it now describes Figure B.

Step 3. Make up a description D for the way that DA was changed in Step 2.

Step 4. Use D TO CHANGE DC. If the resulting description describes one of the answer choices much better than any of the others, we have our answer. Otherwise, start over, but next time use different descriptions for DA, DC and (perhaps) for D.

Notice that Step 3 asks for a description at a higher level! The descriptions in Steps 1 and 2 describe pictures, e.g., "There is a square below a circle." The description in Step 3 describes changes in descriptions, e.g., "The things around the upper figure in DA is around the lower figure in DB." Our thesis is that one needs both of these kinds of description-handling mechanisms to solve even simple problems of vision. And once we have such mechanisms, we can easily solve not only harder visual problems but we can adapt them to use in other kinds of intellectual problems as well -- for learning, for language, and even for kinesthetic coordination.

This schematic plan was the main idea behind a computer program written in 1964 by T. G. Evans. Its performance on "standard" geometric analogy tests was comparable to that of fifteen-year old children! This came as a great surprise to many people, who had assumed that any such "mini-theory" would be so extreme an oversimplification that no such scheme could approach the complexity of human performance. But experiment does not bear out this impression. To be sure, Evans' program could handle only a certain kind of problem, and it does not become better at it with experience. Certainly, we cannot propose it as a complete model of "general intelligence". Nonetheless, analogical thinking is a vital component of thinking, hence having this theory (Evans, 1964), or some equivalent, is a necessary and important step.

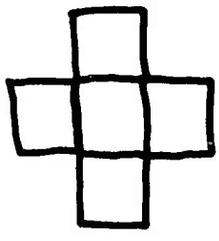
In developing our simple schematic outline into a concrete and complete computer program, one has to fill in a great deal of detail: one must decide on ways to describe the pictures, ways to change descriptions, and ways to describe those changes. One also has to define a policy for deciding when one description "fits much better" than another. One might fear that the possible variety of plausible descriptions is simply too huge to deal with; how can we decide which primitive terms and relations should be used? This is not really a serious problem. Try, yourself, to make a great many descriptions of the relation between A and B that might be plausible (given the limited resources of a child) and you will see that it is hard to get beyond simple combinations of a few phrases like "inside of", "left of", "bigger than", "mirror-image of", and so on.

But let us postpone details of how this might be done (see Evans, 1964) and continue to develop our central thesis: by operating on descriptions (instead of on the things themselves), we can bring many problems that seem at first impossibly non-mechanical into the domain of ordinary computational processes.

What do we mean by "description"? We do not mean to suggest that our descriptions must be made of strings of ordinary-language words (although they might be). The simplest kind of description is a structure in which some features of a situation are represented by single ("primitive") symbols, and relations between those features are represented by other symbols -- or by other features of the way the description is put together. Thus, the description is itself a MODEL -- not merely a name -- in which some features and relations of an object or situation are represented explicitly, some implicitly, and some not at all. Detailed examples are presented in 4.3 for pictures, and in 5.5 for verbal descriptions of physical situations. In 5.6 there are some descriptions which resemble computer programs. If we were to elaborate our thesis in full detail we would put much more emphasis on procedural (program-like) descriptions because we believe that these are the most useful and versatile in mental processes.

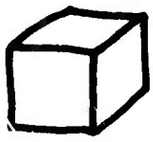
1.2 Children's Use of Descriptions

The theory of analogy we have just proposed might seem both too simpleminded and too abstract to be plausible as a theory of how humans make analogies. But there is other evidence for the idea that mental visual images are descriptive rather than iconic. Paradoxically, it seems that even young children (who might be expected to be less abstract or formal than adults) use highly schematic descriptions to represent geometric information.



We asked a little boy of 5 years to draw a cube. This is what he drew. "Very good," we said, and asked: "How many sides has a cube?" "Four, of course," he said.

"Of course," we agreed, recognizing that he had understood the ordinary meaning of "side", as of a box, rather than the mathematical sense in which top and bottom have no special status. "How many boards to make a whole cube, then?" "Six," he said, after some thought. We asked how many he had drawn. "Five." "Why?" "Oh, you can't see the other one!"



Then we drew our own conventional "isometric" representation of a cube. We asked his opinion of it. "It's no good." "Why not?" "Cubes aren't slanted!"

Let us try to appreciate his side of the argument by considering the relative merits of his "construction-paper" cube against the perspective drawing that adults usually prefer. We conjecture that, in his mind, the central square face of the child's drawing, and the four vertexes around it, are supposed in some sense to be "typical" of all the faces of the cube. Let us list some of the properties of a real three-dimensional cube:

Each face is a square.
Each face meets four others.
All plane angles are right angles.
Each vertex meets 3 faces.
Opposite edges on faces are parallel.
All trihedral angles are right angles, etc.

Now, how well are these properties realized in the child's picture?

Each face is a square.
The "typical" face meets four others!
All angles are right!
Each typical vertex meets 3 faces.
Opposite face edges are parallel!
There are 3 right angles at each vertex!

But in the grown-up's pseudo-perspective picture we find that:

Only the "typical" face is square.
Each face meets only two others.
Most angles are not right.
One trihedral angle is represented correctly in its topology, but only one of its angles is right.
Opposite edges are parallel but only in "isometric", not in true perspective.

And so on. In the balance, one has to agree that the geometric properties of the cube are better depicted in the child's drawing than in the adult's! Or, perhaps, one should say that the properties depicted symbolically in the child's drawing are more directly useful, without the intervention of a great deal more knowledge.

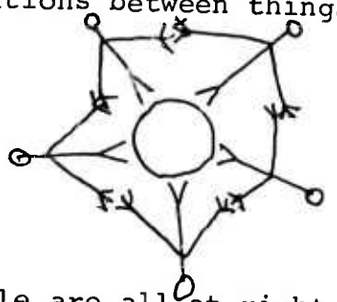
One could argue that in the adult's drawing, the square face and the central vertex are understood to be "typical". We

gave him the benefit of the doubt. Also, one never sees more than 3 sides of a cube, but children can't seem to know this, or feel that it is important. The parallelisms and the general "four-ness" surely dominate.

Incidentally, we do not mean to suggest that our child had in his mind anything like the graphical image of his drawing, but rather that he has a structural network of properties, features, and relations of aspects of the cube, and that what he drew matches this structure better than does the adult's more iconic picture. In 4.4 we will show how such structural networks can be used as a program that learns new concepts as a result of experience.

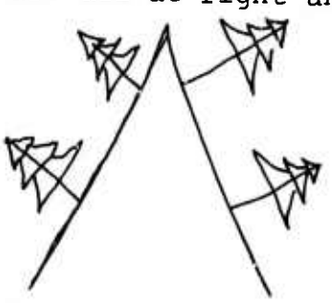
Not all children will draw a cube just this way. They usually draw some arrangement of squares, however, and this sort of representation is typical of children's drawings, which really are not "pictures" at all, but attempts to set down graphically what they feel are the important relations between things and their parts.

Thus "a ring of children holding hands around the pond" is drawn like this, perhaps because the correct perspective view would put some of the children in the water.



Also, in the child's drawing the people are all at right angles to the ground, as they should be!

For the same reason, perhaps, "trees on the mountain" is drawn this way because trees usually grow straight out of the ground. It doesn't matter if an actual scene is right in front of the child; he will still draw the trees sideways!



A person is often drawn this way, perhaps partly because the body that is so important to the adult doesn't really do much for the child except get in his way, partly because it does not have an easily-described shape.

From all this we are led to a new view of what children's drawings mean. The child is not trying to draw "the thing itself" -- he is trying to make a drawing whose description is close to his description of that thing -- or, perhaps, is constructed in accord with that description. Thus the drawing problem and the analogy problem are related.

We hope no reader will be offended by the schematic simplicity of our discussion of "typical children's drawings". Certainly

we are focusing on some common phenomena, and neglecting the fantastic variety and plasticity of what children do and learn. Yet even in that plasticity we see the dominance of symbolic description over iconic imitation.

Most children before 5 or 6 years old draw people like this. Find such a child and ask him, "Where is his hair?" and draw some, or say, "Why doesn't his nose stick out?" and draw an angular line in the middle of the face.

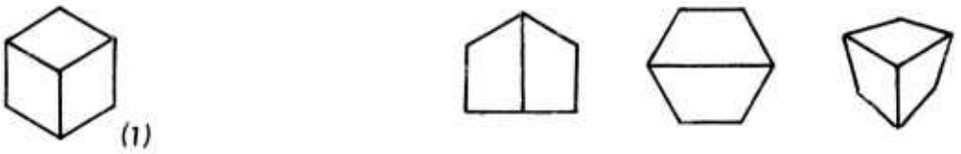
Chances are that if the child pays any attention at all and likes your idea, these features will appear in every face he draws for the next few months. The hair is obviously symbolic. The new nose is no better, optically, than the old, but the child is delighted to learn a symbolism to depict protusion.

There is a vast literature describing phenomena and theories of "learning" in terms of the gradual modification of behavior (or behavioral "dispositions") over long sequences of repetition and tedious "schedules" of reward, deprivation and punishment. There is only a minute amount of attention to the kind of "one-trail" experience in which you tell a child something, or in which he asks you what some word means. If you tell a child, just once, that the elephants in Brazil have two trunks, and meet him again a year later, he may tell you indignantly that they do not.

The success of Evans' program for solving analogy problems does not prove anything, in a strict sense, about the mechanisms of human intelligence. But such programs certainly do provide the simplest (indeed, today the only) models of this kind of thinking that work well enough to justify serious study.

It is natural to ask whether human brains "really" use symbolic descriptions or, instead, manage somehow to work more "directly" with something closer to the original optical image. It would be hard to design any direct experiment to decide such a question in view of today's limited understanding of how brains work. Nevertheless, the formalistic tendencies shown in the children's drawings point clearly toward the symbolic side. The phenomena in the drawings suggest that they are based on a rather small variety of elementary object-symbols, positioned in accord with a few kinds of relations involving those symbols, perhaps taken only one or two at a time. These phenomena are not seen so clearly in the pictures of sophisticated artists, but even so we think the difference is only a matter of degree. While it is possible to train oneself to draw with quantitative accuracy, some aspects of the "true" visual image, the very difficulty of learning this is itself an indicator that the symbolic mode is the more normal manner of performance. Even

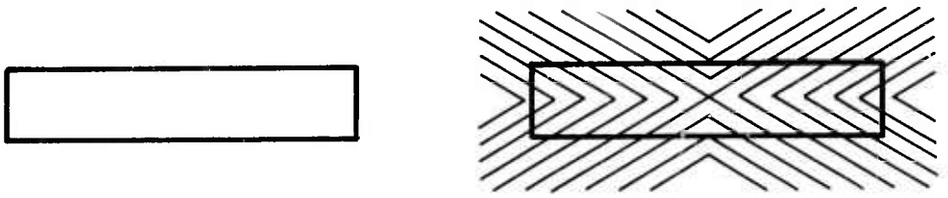
sophisticated adults often show a preference for unreal but tidy "isometric" drawings over more "realistic" perspective drawings:



even though a cube is never seen exactly as in (1). In any case, all this suggests that "graphic" visual mechanisms become operative later (if at all) in human intellectual development than do methods based on structural descriptions. This conclusion seems surprising because in our culture we are prone to think of symbolic description as advanced, abstract, and intellectual, hence characteristic of more advanced stages of maturation.

2.1 Appearance and Illusion

Now consider some phenomena that might seem to be more visual, less intellectual. These two figures show the same rectangle.

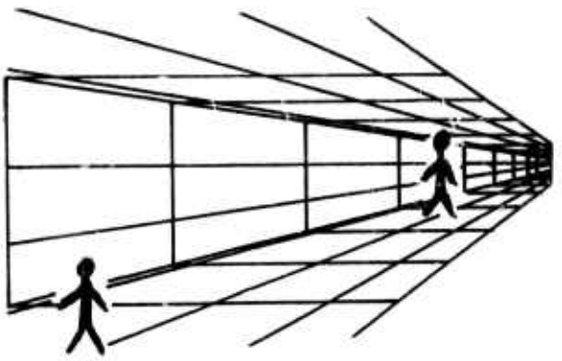


But on the right, the diagonal stripes affect its appearance so that (to most people) the sides appear to lean out and no longer seem perfectly parallel. Such phenomena have been studies with great intensity by psychologists. In the next two figures, the central squares actually have the same grey color, but everyone sees the one at the left as darker.

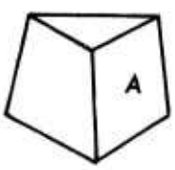
A good deal is known about the effects of nearby figures or backgrounds on another figure. Perhaps most familiar is the phenomenon in which the directions of the oblique segments make the horizontal line in the left figure appear shorter than that in the right figure.

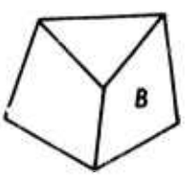
But the strangest illusion of all is this: to many psychologists these phenomena of small perceptual distortions have come to seem more important than the question of why we see the figures at all, as "rectangle", or "square", or as "double-headed arrows!". Surely this problem of how we analyze scenes familiar objects is a more central issue.

Thus one finds much more discussion why the smaller figure looks larger in pictures like this than about why one sees the figures as people at all.



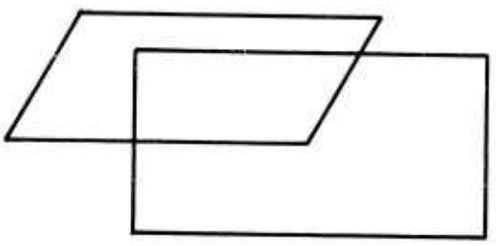
We agree that the study of distortions, ambiguities, and other "illusions" can give valuable clues about visual and other mechanisms. To resolve two or more competing theories of vision, such evidence might become particularly useful. First, however, we need to develop at least one satisfactory theory of how "normal" visual problems might be handled, particularly scenes that are complicated but not especially pathological.





Let us look at a few more visual phenomena. Both of these figures appear at first sight to be reasonable pictures of pyramid-bases -- that is, of simple flat-surfaced, five-faced bodies that could be pyramids with their tops cut off. But in fact, Figure B cannot be a picture of such a body. For its three ascending edges (if extended) would not meet at a single point, whereas those of Figure A do form a vertex for a pyramid.

So here we have a sort of negative illusion; Figure B would not "match" a real photograph of any pyramid-vase. However, it could match quite well an abstract description of a pyramid base -- say, one that describes how its faces and edges fit together (qualitatively, but not quantitatively).



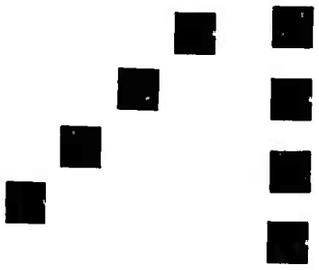
Another topic concerns "camouflaged" figures. The figure "4" embedded in this drawing is not normally seen as such because, we presume, one describes the scene as a square and parallelogram.

Study of this kind of concealment can tell us something about the "principles" according to which our visual system "usually" describes scenes as made up of objects. But once the "4" has been pointed out or discovered, it is then "seen" quite clearly! A good theory must also account for phenomena in which it is

possible to change and elaborate one's "image" of the same scene in ways that depend on changes in his interpretation and understanding of the structure "shown" in the picture.

A simpler -- and more interesting -- example of a figure with two competitive descriptions is the ordinary square! Young children know the square and the diamond as two quite distinct shapes, and the ambiguity persists in adults, as seen here. (See Attneave, 1968.)

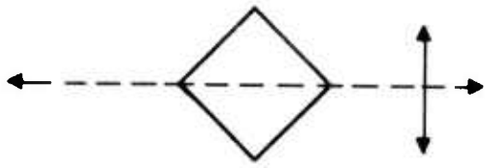
The four objects at the left are usually seen as diamonds, while those on the right are seen as squares. How can we explain this? Since the individual objects are in fact identical, the effect must have something to do with their arrangement. It is tempting to incant the phrase -- "the whole is more than the sum of the parts".

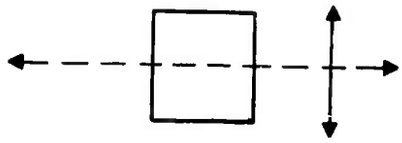


Now consider a descriptive theory. If one is asked to describe this scene, he will say something like: "There are two rows, each with four objects. One is a horizontal row of -- etc." We ignore details here, but suggest that the description is dominated by the grouping into rows, as indicated by their priority in the verbal presentation of the description. In Section 4.6 we discuss a program that does something of this sort.

> By "description" we do not usually mean "verbal description"; we mean an abstract data structure in which are represented features, relations, functions, references to processes, and other information. Besides representing things and relations between things, descriptions often contain information about the relative importance of features to one another, e.g., commitments about which features are to be regarded as essential and which are merely ornamental. For example, much of linguistic structure is concerned with the ability to embed hierarchies of detail into descriptions: subordinate clause formation and other word-order choices often reflect priorities and progressions of structural detail in the descriptions that are "meant". We will return to this in Section 5.

Once commited to describing a row of things, the choice between seeing squares and diamonds begins to make more sense. Which description does one choose? Apparently, the way one describes a square figure depends very much on how one chooses (in one's mind) the axis of symmetry. Consider the differences in the figures' descriptions in each of the two obvious choices of orientation shown in the next figure.

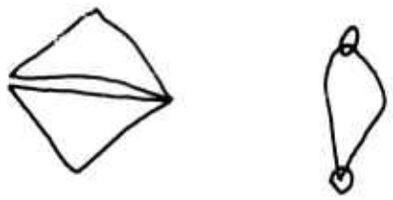




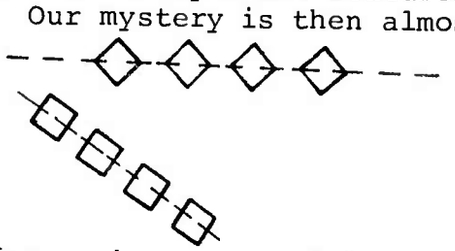
points on axis
one point on each side
made of two triangles
unstable on ground
hurts when squeezed

sides parallel to axis
two points on each side
made of two rectangles
stable -- flat bottom
safe to pick up

These two descriptions could hardly be more different! No wonder that most 3 year olds do not believe that they are the same. In fact, children's drawings of diamonds often come out as



indicating that their descriptive image is a composition of two triangles, or at least that the most important features are the points on the symmetry axes. Our mystery is then almost solved: whatever process set up the description in terms of rows set up also a spatial frame of reference for each group.



Since one has to choose an axis for each square and "other things being equal" there is no strong reason locally for either choice, one tends to use the axis inherited from the direction of its "row". The fact that you can, if you want, choose to see any of the objects as either diamond or square only confirms this theoretical suggestion -- the choice is by default only, and hence would be expected to carry little force.

Once this door is opened, it suggests that other choices one has to make in visual description also can depend on other alien elements in one's thoughts -- as well as on other things in the picture! Every simple figure is highly ambiguous. In a face, a circle can be an eye, a mouth, an ear, or the whole head. There should be no difficulty in admitting this to our theory -- or to the computer programs that demonstrate its consistency and performance. Traditional theories directed toward physical (rather than on computational, or symbolic) mechanisms were inherently unable to account for the influence of other knowledge and ideas upon "perception".

2.2 Sensation, Perception and Cognition

Our discussion of how images depend on states of mind is part of a broader attack on the conventional view of the structure of mind. In today's culture we grow up to believe that mental activity operates according to some scheme in which information is transformed through a sequence of stages like:

Although it is hard to explain exactly what these stages or levels are, everyone comes to believe that they exist. The "new look" in ideas about thinking rejects the idea that there are separate activities like "perception" that precede and are basically independent of "higher" intellectual activities. What one "sees" depends very much on one's current motives, intentions, memories, and acquired processes. We do not mean to say either that the old layer-cake scheme is entirely wrong or that it is useless. Rather, it represents an early concept that was once a clarification but is now a source of obscurity, for it is technically inadequate against the background of today's more intricate and ambitious ideas about mechanisms.

> The higher nervous system is embryologically, and anatomically divided into stages of some sort and this might suggest a basis for the popular-science hierarchy. This makes sense for the most peripheral sensory and motor systems, in which transmission between anatomical stages is chiefly unidirectional. But (presumably) when we go further in the central direction this is no longer true, and one should not expect the geometrical parts of a cybernetic machine to correspond very well to its "computational parts".

Indeed, the very concept of "part", as in a machine, must be rebuilt when discussing programs and processes. For example, it is quite common in computer programs -- and, we presume, in thought processes -- to find that two different procedures use each other as subprocedures! We shall see this happening throughout Section 5. In such a case, one can hardly think of either process as a proper part of the other. So the traditional view of a mechanism as a HIERARCHY of parts, subassemblies and sub-sub-assemblies (e.g., the main bearing of the fuel pump of the pitch vernier rocket of the second ascent stage) must give way to a HETERARCHY of computational ingredients.

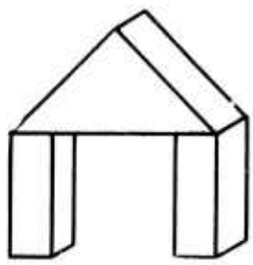
It is unfortunate that technical theories, and even practical guidelines, for such heterarchies are still in their infancies. The rest of this chapter discusses some aspects of this problem.

2.3 Parts and Wholes

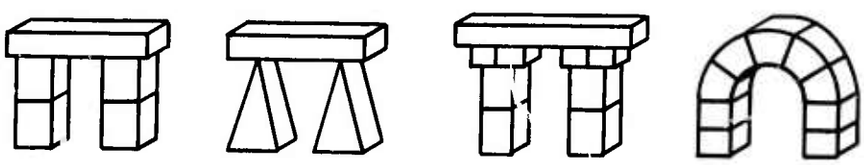
A recurrent theme in the history of psychological thinking involves recognizing an important distinction without having the

technical means to give it the appropriate degree of precision. Consequently, the dividing line becomes prematurely entrenched in the wrong place. An influential example was the concept of "Gestalt". This word is used in attempts to differentiate between the simplest immediate and local effects of stimuli, and those effects that depend on a much more "global" influence of the whole stimulus "field".

Here is a visual example in which this kind of distinction might be considered to operate: In one sense, this arch is "nothing but" three blocks.

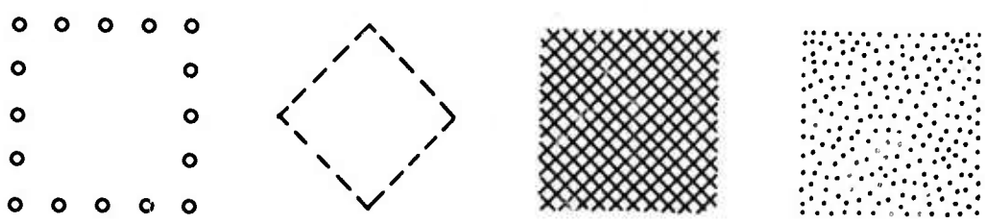


But the arch has properties -- as a single whole -- that are not inherited directly from properties of its parts in any simple way. Some of those arch properties are shared also by these structures:



MORE ARCHES

Obviously the properties one has in mind do not reside in the individual building blocks, they "emerge" from the arrangements of those parts. And one finds this in even simpler situations. Obviously we react to a simple outline square in a way that is very different from our reactions to four separate lines, and rather similar to how we react to such graphically different figures as these:

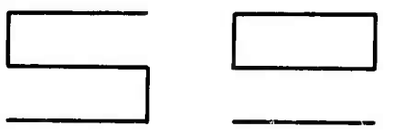


SQUARES

The question "whence comes the square if not from its parts" is not really very serious here, for it is easy to make theories about how one might "perceive" a shape if there are enough easily-detected features to approximately delineate its

geometric form. But there is no similarly easy solution to the kinds of problems that arise when one looks at three-dimensional scenes.

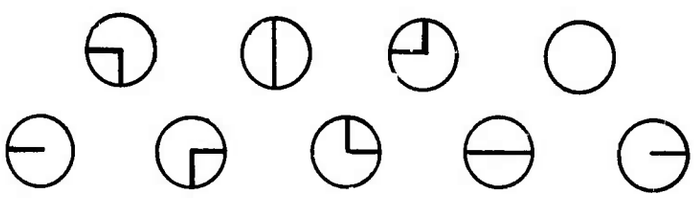
The next two figures are "locally identical" in the following precise sense: Imagine innumerable experiments, in each of which we choose a different point of the picture to look at, and record what we see only within a very small circle around that point.



CONNECTED DISCONNECTED

Both pictures would produce identical collections of data! -- provided that we keep no records of the locations of the viewpoints. So in this sense both pictures have the same "parts". They are obviously very different, however.

One particularly outstanding difference is that one picture is all in one piece -- it is CONNECTED -- while the other is not. In fact, both pictures are composed of just these kinds of "micro-scenes":



In our book Perceptrons we prove that,in general, one cannot use statistics about such local evidence to distinguish between figures that are "connected" and those that are not.

From this, one might conclude that one can tell very little about a picture from such "spatially local" evidence. But this is not true. For example, we can completely define the property of being "m e-entirely-of-separate,-solid,-rectangles" by requiring that all very small parts of the scene look like one or another of these micro-scenes:

that is, every micro-scene must be either homogeneous, a simple edge, or a convex right-angle corner.

It is not hard to see that this definition will accept any picture that contains only solid rectangles, but no other kind of picture. So in this sense, "rectangle-ness" can be defined in terms of local properties, while connectedness cannot. Try to define, "composed-of-a-single-solid-rectangle" in this way. It cannot be done! So we see a difference between two kinds of categories of pictures, in regard to the relations between their parts and their wholes!

The question, "Is the whole more than the sum of its parts?" is certainly provocative and insightful. But it must be recognized also as vague, relative, and metaphorical. What is meant by "parts" and, more important, what is meant by "sum"?

In the case of the rectangles a trivial sense of "sum" will suffice: not even adding up evidence is necessary, for we can make the decision in favor of rectangle, and let any single exception to our condition on the local "micro-scenes" have absolute veto power. So the"sum of the parts" is simply the agreement of all local evidence. For connectedness we seem to need something more complicated, computationally. We have studied this situation rather deeply in Perceptrons: connectedness is a property that is quite important and very thoroughly understood in classical mathematics; it is in fact the central concern of the entire subject of Topology.

For example, here are several quite different-looking conditions,each of which can be used to define the same concept of connectedness:

> PATH-CONNECTION. For any two black points of the picture, there is a path connecting them that lies entirely in black points.
>
> PATH-SEPARATION. There is no closed path, entirely in white points, such that there are some black points inside the path and some black points outside the path.
>
> SET-SEPARATION. The black points cannot be divided into two non-empty sets which are separated by a non-zero distance -- that is, no pair of points, one from each set, are closer than a certain distance.
>
> TOTAL-CURVATURE. Assume that there are no "holes" in the black set -- that is, white points that are cut off from the outside by a barrier of black points. Then compute the sum of all the boundary curvatures (direction-changes at all edges of the figure), taking convex curves as positive and concave curves as negative. The picture is connected if this sum is exactly 360 degrees. If it is a multiple of 360, this gives the number of objects!

Each of these suggests different computational approaches. Depending upon what resources are available, one or another will

be more efficient, use more or less memory, time, hardware, etc. Each definition involves very large calculations in any case, except the fourth, in which one computes simply a sum of what one observes in each small neighborhood. However, the fourth definition does not work in general, but only for figures without holes. And, to be sure that condition is satisfied one must have another source of information (e.g., if one knows he is counting pennies) or else the definition is somewhat circular, because to be able to see that there are no holes is really equivalent to being able to see that the background is connected!

We know exactly what it means for the number seven to be the sum of the numbers three and four. But when we ask whether a house is just the sum of its bricks, we are in a more complicated situation. One might answer:

> "Yes, there is nothing but bricks there."

But another kind of answer could be

> "No, for the same bricks arranged differently would have made a very different house."

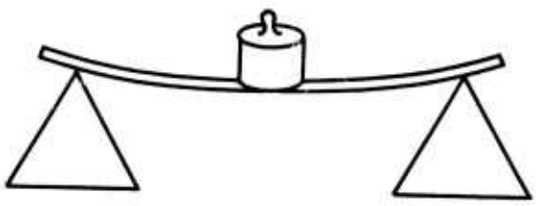
The answer must depend on the purpose of the question. If we admit only "yes" or "no", there is no room for refinement and subtlety of discussion. We do not really want either of the answers "Yes, it is nothing but the sum" or "No, it is a Gestalt, a totally different and new thing". We really want to know exactly how the response, image, or interpretation of the situation is produced: we want an explanation of the phenomenon. And the terms of the explanation must be appropriate to the kind of technical question we have in mind. Sometimes one wants the result in terms of a particular set of psychological concepts, sometimes in terms of the interconnections of some perhaps hypothetical neural pathways, and sometimes in terms of some purely computational schemata.

Thus one might ask, about some aspect of a person's behavior:

> COMPONENTS: Can the phenomenon be produced in a certain kind of theoretical neural network?
>
> LEARNING: Can it be learned by a certain kind of reinforcement schedule according to certain proposed laws of conditioning?
>
> COMPUTATIONAL STRUCTURE: Can this result be computed by a computer-like system subject to certain restrictions, say, on the amount of memory, or on the exclusion of certain kinds of loops interconnecting its components?
>
> COMPUTATIONAL SCHEMATA: Can the outer behavior of this individual reasonably be imitated by a program containing such-and-such a data-structure and such-and-such a syntactic analyser and synthesizer?

The way in which the whole depends upon its parts, for any phenomenon, has a direct bearing on how such questions can be answered. But to supply sensible answers, one needs a stock of crisp, precise, ideas about how parts and wholes may be related!

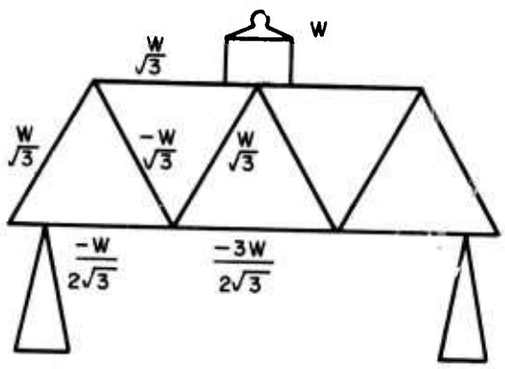
It is important to recognize that these kinds of problems are not special to psychology. Water has properties that are not properties either of hydrogen or oxygen, yet chemistry is no longer plagued by fights between two camps -- say, "Atomist" vs. "Gestalt". This is not at all because the problem is unimportant: exactly the opposite! The reason there are no longer two camps in chemistry is because all workers recognize that the central problems of the field lie in developing good theories of the different kinds of interactions involved, and that the solution of such problems lie in constructing adequate scientific and mathematical models rather than in defending romantic but irrelevant philosophical overviews. But in psychology and biology, there remains a widespread belief that there are phenomena of mind or of cell that are not "reducible" to properties and interactions of the parts. They are saying, in essence, that there can be no adequate theory of the interactions.



SUPPORTED ROD

Consider a concrete example. It is relatively easy to bend a thin rod, but much harder to bend this structure made of several such rods. Where does the extra stiffness come from?

The answer, in this case, is that the "new property" is indeed inherited from the parts, because of the arrangement, but in a peculiar way. In the truss, a force at the middle is resisted -- not by bending-forces across the rods -- but by compression and tension forces along the rods.



TRUSS

The resistance of a thin rod to forces along it is much greater than the resistance to forces across it. So the increases strength is indeed "reduced", in the Theory of Static Mechanics, to the interactions of stresses between members of the structure. Even the properties of a single rod itself can be explained in terms of more microscopic interactions of the tensile and compressive forces between its own (!) "parts", when it is strained. By imagining the rod itself to be a truss (a heuristic planning step that helps one to write down the correct differential equation) we can analyze stress-strain relations

inside the rod. Thus one obtains such a beautiful and accurate model that there remains no mysterious "Gestalt" problem at all.

This is not to say that special arrangements have no special properties. In some of Buckminster Fuller's work, the dodecahedral sphere yields a kind of structural stiffness rather different than that in the triangular truss. Here the rigidity does not come directly from that of small or "local" triangular substructures, and it takes a different kind of mathematical analysis to see why it is hard to distort it. Even so, there remains no mysterious "emergent" property here that cannot be deduced from the classical theory of statics.

Of course, our real concern is with problems of intelligence, rather than with engineering mechanics. But many problems that seem at first to be "purely psychological" often turn out to center around just such problems of wholes and parts. And with such an interpretation, we may replace an elusively ill-defined psychological puzzle by a much sharper problem within the theory of computation.

The computer is the example par excellence of mechanisms in which one gets complex results from simple interactions of simple components. In asking how thought-like activity could be embedded in computer programs, scientists for the first time really came to grips with understanding how intelligent behavior could be made to emerge from simple interactions.

The issue seems really to be fundamentally one of assessing the complexity of processes. The content of the Gestalt discoveries is that certain psychological phenomena require forms of computation that lie outside the scopes of certain models of the brain -- and outside certain conjectures about the "elementary" units of which behavior is supposed to be composed. So, the whole discussion must be considered in relation to some overt or covert committment about what units of behavior, or of brain-anatomy, or of computational capacity, are supposed to be "atomic".

To illustrate extreme versions of atomism vs. Gestaltism one might consider these caricatures:

> EXTREME ATOMISM: All behavior can be understood in terms of simple functions of neural paths that run from single receptors, through internuncials, to effectors.
>
> EXTREME GESTALTISM: The essence in this is the whole pattern. Many simple examples show that the response is made to the whole stimulus and cannot be represented as simple sums or products of simple local stimulations.

Clearly one does not want to set a threshold between these; one wants to classify intermediate varieties of interactions that might be involved, arranged if possible in some natural order of complexity.

Thus in Perceptrons we studied a variety of simple schemas such as these:

> EXTREMELY ATOMIC ALGORITHM: One of the input wires is connected to the output, the others to nothing.
>
> VETO ALGORITHM: If every input says "yes", the output is "yes". If any input says "no", the output is "no".
>
> MAJORITY ALGORITHM: If M or more of N inputs say "yes", output is "yes".
>
> LINEAR SUM ALGORITHM: To each input is assigned a "weight". Add together the weights for just those inputs that say "yes". The output is just this sum.
>
> LINEAR THRESHOLD ALGORITHM: Use the LINEAR SUM algorithm, except, make the output "yes" if the sum is greater than a certain "threshold", otherwise the output is "no".

Exercise: the reader should convince himself that "extremely atomic", "veto", and "majority" are special cases of "linear threshold".

> EQUIVALENT-PAIR ALGORITHM: The input is considered to be grouped in pairs. The output is "yes" only when, for every pair, the two members have the same input values.

The reader should convince himself that this is not a special case of "linear threshold"!

> SYMMETRICAL ALGORITHM: The response is "yes" if the pattern of inputs is symmetrical about some particular center, or about some particular linear axis.

This is a special case of the equivalent-pair algorithm. They are both examples of perceptrons in which the global function can be expressed as a linear threshold function of intermediate functions of two variables. Here the whole is only trivially more than the sum of the parts.

> PERCEPTRON ALGORITHM: First some computationally very simple functions of the inputs are computed, then one applies a linear threshold algorithm to the values of these functions.

Many different classes of perceptrons have been studied; such a class is defined by choosing a meaning for the phrase "very simple function". For example, one might specify that such a function can depend on no more than five of the stimulus points. This would result in what is called an order-five perceptron. All of the examples above had order one or two. The next example has no "order restriction", but the functions

are very simple in another sense; they are themselves "order one" or linear-threshold functions.

> GAMBA PERCEPTRON: A number of linear threshold systems have their outputs connected to the inputs of a linear threshold system. Thus we have a linear threshold function of many linear threshold functions.

Virtually nothing is known about the computational capabilities of this latter kind of machine. We believe that it can do little more than can a low order perceptron. (This, in turn, would mean, roughly, that although they could recognize some relations between the points of a picture, they could not handle relations between such relations to any significant extent.) That we cannot understand mathematically the Gamba perceptron very well is, we feel, symptomatic of the early state of development of elementary computational theories.

Which of these are atomic and which Gestaltist? Rather than muddle through a philosophical discussion of which cases "really" do more than add the parts, we should try to classify the kinds of mechanisms needed to realize each in certain "hardware" frameworks, chosen for good mathematical reasons. Then for each such framework, we might try to see which admit simple reinforcement mechanisms for learning, which admit efficient descriptive teaching (see Section 4), which admit the possibility of the cognitive machinery "figuring out for itself" what are the important aspects of a situation!

To supply such ideas, we have to make theoretical models and systems. One should not expect to handle complex systems until one thoroughly understands the phenomena that may emerge from their simpler subsystems. This is why we focused so much attention on the behavior of perceptrons in problems of computational geometry. It is important to emphasize that we want to understand such systems for the reasons explained above, rather than as possible mechanisms for practical use. When a mathematical psychologist uses terms like "linear", "independent", or "Markoff Process", etc., he is not (we hope!) proposing that a human memory is one of those things; he is using it as part of a well-developed technical vocabulary for describing the structure of more complicated schemata. But until recently there was a serious shortage of ways to describe more procedural aspects of behavior.

The community of ideas in the area of computer science makes a real change in the range of available concepts. Before this, we had too feeble a family of concepts to support effective theories of intelligence, learning, and development. Neither the finite-state and stimulus-response catalogs of the Behaviorists, the hydraulic and economic analogies of the Freudians, or the holistic insights of the Gestaltists supplied enough technical ingredients to develop such an intricate subject. It needs a substrate of debugged theories and solutions to related but simpler problems. Computer science has brought a flood of such ideas, well defined and experimentally implemented, for thinking about thinking; only a fraction of them have distin-

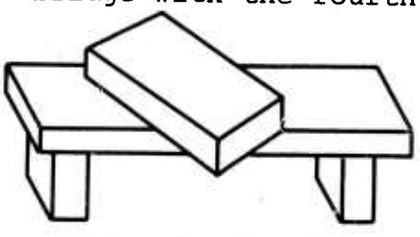
guishable representations in traditional psychology:

symbol table	closed subroutine
pure procedure	pushdown list
time-sharing	interrupt
calling sequence	communication cell
functional argument	common storage
memory protection	decision tree
dispatch table	hardware-software trade-off
error message	serial-parallel trade-off
function-call trace	time-memory trade-off
breakpoint	conditional breakpoint
formal language	asynchronous processing
compiler	interpreter
indirect address	garbage collection
macro language	list structure
property list	block structure
data type	look-ahead
hash coding	look-behind (cache)
micro-program	diagnostic program
format matching	executive program
syntax-direction	operating system

These are only a few ideas from the environment of general "systems programming" and debugging; we have mentioned none of the much larger set of concepts specifically relevant to programming languages, artificial intelligence research, computer hardware and design, or other advanced and specialized areas. All these serve today as tools of a curious and intricate craft, programming. But just as astronomy succeeded astrology, following Kepler's discovery of planetary regularities, the discoveries of these many principles in empirical explorations of intellectual processes in machines should lead to a science, eventually.

3. Analysis of Visual Scenes

No one could have any doubt about what this picture is supposed to show: "Four blocks, three forming a bridge with the fourth lying across it." We would like to program a machine to be able to understand scenes to at least this level of comprehension. Notice that our description involves recognizing the "bridge" as well as the blocks that comprise it, and that the phrase "lying across it" indicates knowing that the block is actually resting on the bridge. Indeed, in the pronoun reference to the bridge, rather than to the top block of the bridge, there is implied a further level of functional analysis.



FOUR BLOCK BRIDGE

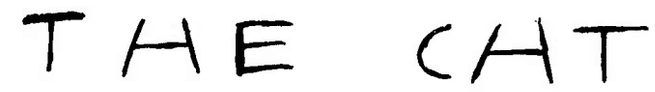
In our earlier progress reports we described the SEE program (Guzman 1968) which was able to assemble the thirty vertices, forty segments and thirteen regions of this picture into four objects, using a variety of relatively local "linkage" cues.

A new program, (Winston 1970) goes further in the analysis of three-dimensional support and can recognize groups of objects as special structures (such as "bridge") to yield just the kind of functional description we are discussing. Winston's program is even able to LEARN to recognize such configurations, using experience with examples and non-examples, as shown in Chapter 4.

Before discussing scene-analysis in detail, we have a few remarks about the nature of problems in this area. In the early days of cybernetics (McCulloch-Pitts 1943, Wiener 1949) it was felt that the hardest problems in apprehending a visual scene were concerned with questions like "why do things look the same when seen from different viewpoints", when their optical images have different sizes and positions.

How does one capture the "abstraction" or "concept" common to all the particular examples. For two-dimensional character-recognition, this kind of problem is usually handled by a two-step process in which the image is first "normalized" to standard position and then "matched" -- by a correlation or filtering process -- to one of a set of standard representatives. In practical engineering applications, the "normalizing" often failed because it could not disarticulate parts of images that touch together, and "matching" often failed because it is hard to make correlation-like processes attend to "important" parts of the figures instead of to ornaments. Even so, such methods work well enough for reasonably standardized symbols.

If, however, one wants the machine to read the full variety of typography that a literate person can, the problem is harder, and if one wants to deal with hand-printing, quite different methods are needed. One is absolutely forced to use exterior knowledge involving the pictures' contexts, in situations like this. (Selfridge, 1955)



Here the distinction between the "H" and the "A" is not geometric at all, but exists only in one's knowledge about the language. An early program that could do this was described in Bledsoe and Browing 1959. But we will not stop to review the field of character-recognition, for its technology is quite alien to the problems of three-dimensional scenes. This is because the problems that concern us most, like how to separate objects that overlap, or how to recognize objects that are partially hidden (either by other objects or by occluding parts of

their own surfaces), simply do not occur at all in the two-dimensional case. Some more interesting two-dimensional problems require description when geometric matching fails; a conceptual "A" is not simply a particular geometric shape; it is

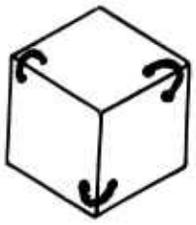
> "Two lines of comparable length that meet at an acute angle, connected near their middles by a third line."

3.1 Programs for Finding Bodies in Scenes

Let us review quickly how Guzman's SEE program works. First a collection of "lower level" programs are made to operate directly on the optical data. Their job is to find geometric features of the picture -- regions, edges and vertices -- so that the scene can be described in a simple way in the program's data-structure. Next, the vertices are classified into "types". The most important kinds are these:

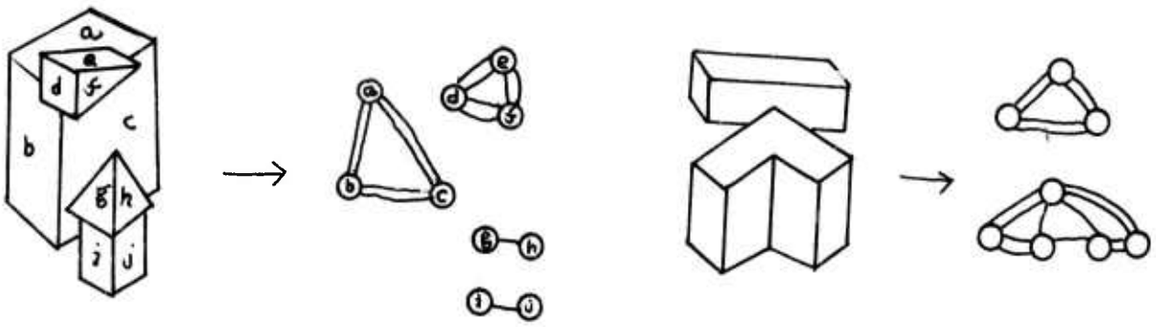
The main goal of the program is to divide the scene into "objects", and its basic method is to group together regions that probably belong to the same object. Each type of vertex is considered to provide some evidence about such groupings, and can be used to create "links" between regions.

For example, the ARROW type of vertex usually is caused by an exterior corner of an object, where two of its plane surfaces form an edge. So we insert a "link" between the two regions that are bounded by the two smaller angles:

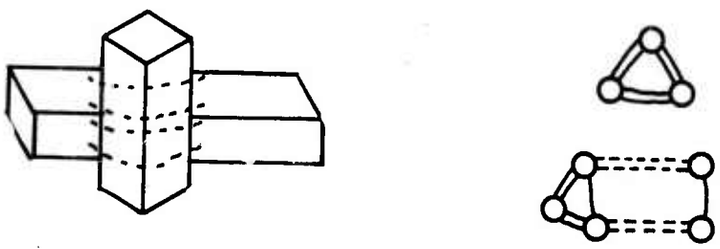


Similarly, the FORK type of vertex, which is usually due to three planes of one object, causes three links between those regions.

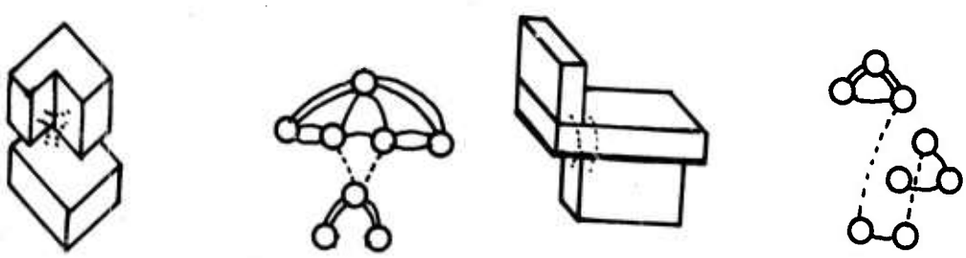
Using these clues, and representing the resulting relations by simple abstract networks, many scenes are "correctly" analyzed into objects.



If two TEE vertices have their stems in the same line then we create two more links: This often does just the right thing for an object whose picture is divided into two separate parts by another object in front.



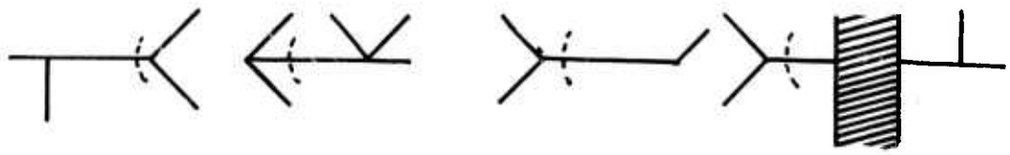
Many scenes are handled correctly by just these simple rules, but many are not. For example, the basic assumption about the FORK linking its three regions is not true of concave corners, and the "matching TEE" assumption may be false by coincidence, so that "false links" may be produced in such cases as these:



Guzman introduced several methods for correcting such errors. One method involves a conservative procedure in which groupings are considered to have different qualities of connectedness. Two high-quality groups that are connected together by only a single link are broken apart -- the link is deleted.

A second error-correction method is more interesting. Here we observe that the TEE vertex really has a special character, quite

opposed to that of the FORK and the ARROW. The most usual physical cause of a TEE is that an edge of one object has disappeared under an edge of another object. Hence, we should regard the TEE joint as evidence against linking the corresponding regions! Guzman's implementation of this was to recognize certain kinds of configurations as special situations in which the existence of one kind of vertex-type causes inhibition or cancellation of a link that would otherwise be produced by the other vertex-type. That would happen, for example, in these figures:

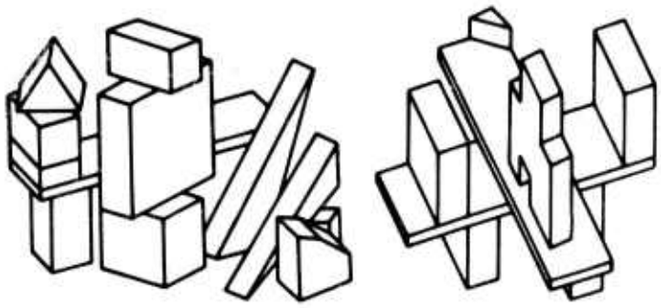


This technique corrects many errors that the more "naive" system makes, especially in objects with concavities. Note that it attempts to compute Connectedness: -- for is not the notion of "object" as we are using it,exactly that idea? -- by extremely local methods, while the (better) system with cancellation is less local because of the effects of vertex-types of contiguous or closely-related geometric features.

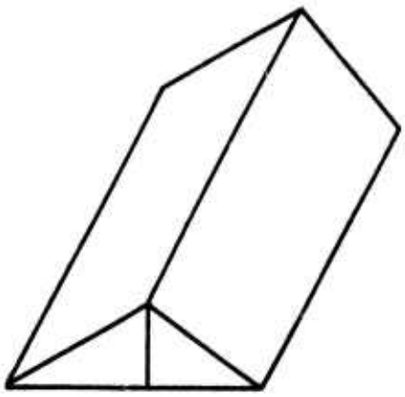
Guzman's method might seem devoid of the normalization and matching operations. Indeed, in a sense it has nothing to do with "recognizing" at all; it is concerned with the separation of bodies rather than with their shapes. But both normalization and matching are more or less inherent in the descriptive language itself, since the very idea of vertex-type is that of a micro-scene which is invariant of orientation, scale, and position. This scheme of Guzman's is very much in accord with the Gestaltists' conceptual scheme in which the separation of figure from background is considered prior to,and more primitive than, the perception of form.

The "cancellation" scheme has a more intelligible physical meaning. It has been pointed out by D. Huffman (1970) that each line in a line-drawing may be interpreted as a physical edge formed (we assume) by the intersection of two planes, at least locally. In some cases, one can see parts of both planes, but in other cases, only one. A T-joint is good evidence that the edge involved is of the latter kind,and once one assigns such an interpretation to an edge, then it follows immediately that the adjacent Guzman links to the alien surface ought to be rejected. Accordingly, Huffman developed a number of procedures for making detailed global interpretations from local edge-region assignments.

We will not give further details of the SEE program here. As an example of its performance, it correctly separates all the objects in this scene.

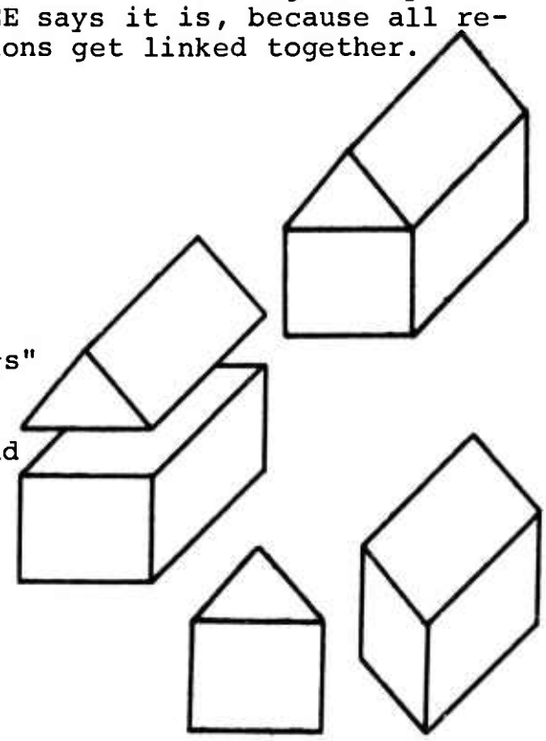


But SEE has faults, among which are:



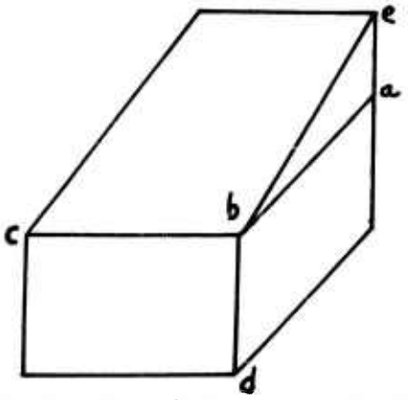
ORDINARY "MISTAKES": Certain simple figures are not handled "correctly". To be sure, all figures are inherently ambiguous (any scene with n regions could conceivably arise from a picture of n objects). Our real goal is to find an analysis that makes sense in everyday situations. Normally one would not suppose that this is a single body, but SEE says it is, because all regions get linked together.

INFLEXIBILITY: If its very first proposal is not acceptable, the body-aggregation program ought to be able to respond to complaints from other higher-lower-level programs and thus generate some alternative "parsings" of the scene. For example, SEE finds a single body in the top one of these figures, but it should be able to produce the two other alternatives shown below it. (It is interesting how difficult it is for some humans to see the third parsing.)



IGNORANCE: It has no way to use knowledge about common or plausible shapes. While it is a virtue to be able to go so far without using such exterior information, it is a fault to insist on this!

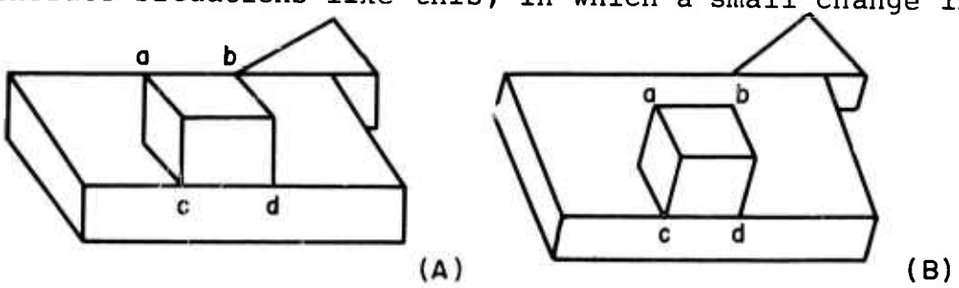
Following Guzman's work, Martin Rattner has described a procedure, called SEEMORE, that can handle some of these problems. [Rattner 1970] While it uses linking heuristics much as did Guzman, SEEMORE puts more emphasis on local evidence that an edge might separate two bodies. These "splitting heuristics" operate initially at certain kinds of vertices, notably TEE-vertices and vertices with more than three edges (which were not much used in earlier programs). When there is more than one plausible alternative, SEEMORE uses other evidence to make tentative choices of how to continue a splitting line, but stores these choices on back-up lists that can later be used to generate alternative parsings.



Here is a simple example. In this figure, one might imagine splitting either along the line a-b-c or along the line d-b-e. The central vertex 'b' suggests (locally) either of these; on the other hand, such splits as a-b-d or a-b-e are considered much less likely.

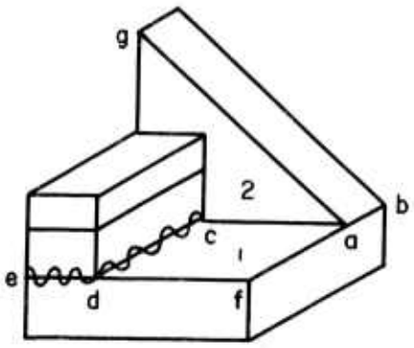
The vertex 'a' strongly suggests a split along a-b, while neither 'c', 'd', nor 'e' have much in their favor. Thus SEEMORE starts a split at 'a' and continues at 'b' toward 'c'. Generally, splits originate at TEE's, propagate through L's and matching TEE's, and avoid the sharpest turns through the multiple-edge vertices.

Degenerate situations like this, in which a small change in



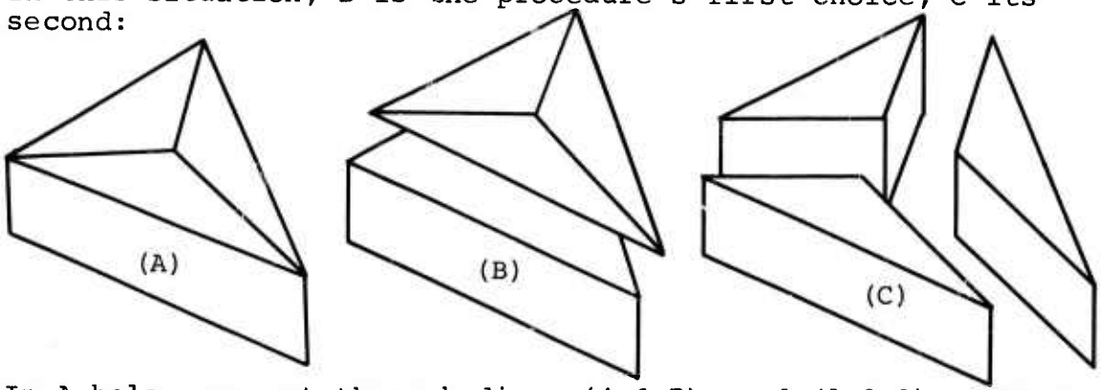
viewing angle produces a different topology, are likely to lead to "incorrect" analyses. Rattner uses a rather conservative linking phase, in which links are placed more cautiously than in SEE, but using similar "inhibiting" rules. Regions that are doubly-linked to one another by these are considered, "strongly" bound; then the heuristic rule is to attempt to split around these "nucleii", and to avoid splitting through them.

It would be tedious to give full details here, partly because the subject is so specialized, but primarily because the procedure has not been tested and debugged in a wide enough variety of situations. A few examples follow.



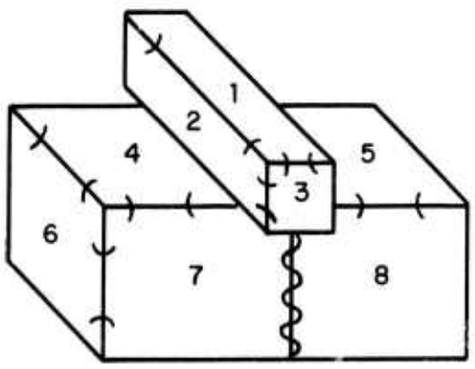
An initial split is made along e-d, extended to d-c. Then, between the possible splits g-a-f and c-a-b, the latter is preferred because it completes the unfinished split ending at 'c'

In this situation, B is the procedure's first choice, C its second:

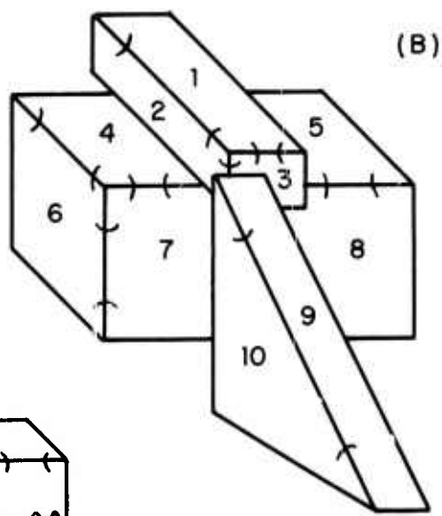


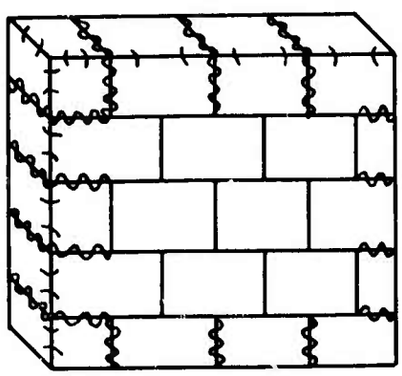
In A below, we get three bodies, (4-6-7), and (1-2-3). SEE does not split between regions 7 and 8. In B, one gets the plausible three-body analysis. If there is any complaint, SEEMORE will propose to separate (4-6-7) and (5-8). In C, all the bricks are properly separated. While SEE would have to put in many spurious links because of the Coincidentally matching TEE's SEEMORE inhibits these on the basis of other splitting evidence.

(A)



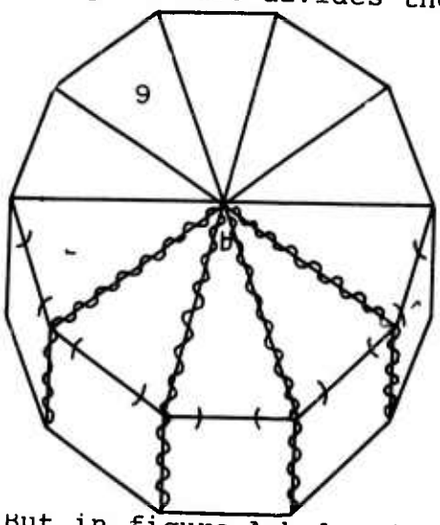
(B)

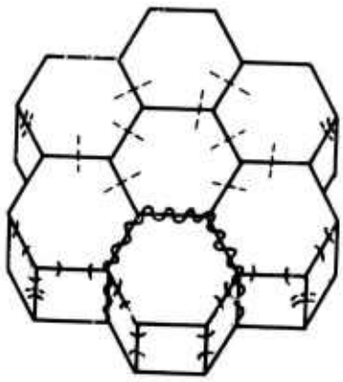




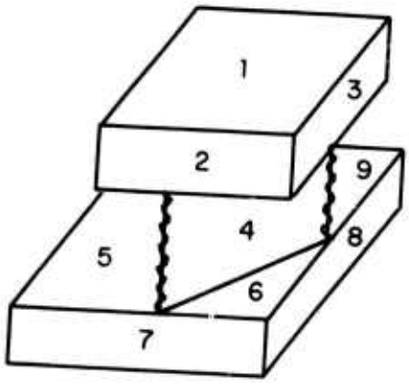
(C)

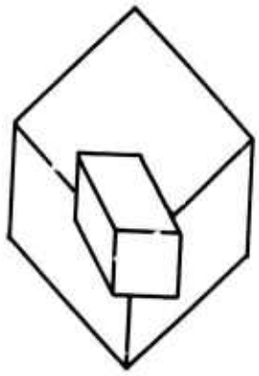
The procedure divides these into the "natural" parts:





But in figure A below it finds three bodies 1-2-3, 5-7-8-9, and 4-6. The latter is perhaps not the first way a person would see it. And the procedure cannot aggregate the outer segments of the larger cube in figure B because its initial grouping process is so conservative. Clearly, such problems eventually must be gathered together in a "commonsense" reasoning system; the multiple T-joints all would meet, if "extended" in such a way as to suggest the proper split, and the program ought to realize this.





4. DESCRIPTION AND LEARNING

The concepts we used to analyze ANALOGY and SEEING are just as vital in understanding LEARNING. It was traditional to try to account for learning in terms of such primitives as "conditioned reflex" or "stimulus-response bond". The phenomena of learning become much more intelligible when seen in terms of "description" and "procedure".

> There might seem a world of difference between activities involving permanent changes in behavior -- and the rest of thinking and problem-solving.

> But even the temporary structures one obviously uses in imagining and understanding have to be set up and maintained for a time. We feel that the differences in degree of permanence are of small importance compared to the problems of deciding what to remember. It is not the details of how recording is done, but the details of how one solves the problem of what to record, that must be understood first.

As we develop this idea, we find ourselves forced to question the whole tradition in which one distinguishes a special sub-set of mental or behavioral processes called "learning". Nothing but disaster can come from looking for three separate theories to explain (for example)

> How one learns mathematics,
> How one thinks mathematically once he has learned to,

and

> What mathematics is, anyway.

We are not alone in trying to replace such subdividions -- but perhaps more radical and thorough-going. In this chapter we shall argue that many problems about "learning" really are concerned with the problem of finding a description that satisfies some goal. Gestalt psychologists also often emphasized the similarity between solving apparently abstract problems and situations that intuitively feel like simple perception; the same relation that is dimly reflected in ordinary language by expressions like

> "I suddently saw the solution!"

We thoroughly agree about bringing these phenomena together, but we have a very different way of dealing with the newly united couple. We might caricature this difference by saying that the Gestaltists might look for simple and fundamental principles about how perception is organized, and then attempt to show how symbolic reasoning can be seen as following the same principles, while we might construct a complex theory of how knowledge is applied to solve intellectual problems and then attempt to show how the symbolic description that is what one "sees" is constructed according to similar such processes. Indeed,we think that ideas that have come from the study of symbolic reasoning have done more to elucidate visual perception than ideas about perception have clarified our thoughts about abstract thinking -- but the whole comparison is too dialectical to try to develop technically.

In any case, we differ from the Gestaltists more deeply in problems of learning, which they neglected almost entirely -- perhaps because that was the favorite subject of the abominable behaviorists! Let us now explain why we feel that learning, technically, cannot usefully be separated from other aspects either of preception or of symbolic reasoning. As usual, we present first a caricature; then point to where the extreme positions might be softened.

Learning -- or "Keeping track"

Everyone would agree that getting to know one's way around a city is "learning". Similarly, we see solving a problem often as getting to know one's way around a "micro-world" in which the problem exists. Think, for example, of what it is like to work on a chess problem (or on a geometry puzzle,or trying to fix something). Here the micro-world consists of the network of situations on the chessboards that arise when one moves the pieces. Solving the chess problem consists largely of getting to know the relations between the pieces, and how the moves affect things. One naturally uses words like "explore" in this context. As the exploring goes on, one experiences events in which one suddenly "sees" certain relations. A grouping first seen as three pieces playing different roles is now described in terms of a single relation between the three, such as "pin", "fork", or "defense". The experience of re-description can be as "vivid" as if the pieces involved suddenly changed color or position.

One might object that the difference between getting to know the city and solving the chess problem is that one remembers the city and forgets the chess situation (assuming that one does). Isn't that what brings one into the domain of learning and excludes the other? Only to a degree! The chess analysis has to be remembered long enough, within the rest of the analysis. To take an extreme form of the argument, one would repeat one's first steps forever unless one remembered which positions had been analyzed, what relations were observed, and how their descriptions were summarized. What is stored within problem-solving is as vital to the immediate solution as what is retained afterwards is to the solution of the presumably larger-scale problems one is embedded in throughout life. Of course there is a problem about how long one retains what one learns -- but perhaps that belongs to the theory of forgetting rather than of learning!

In our laboratory the chess program written by R. Greenblatt plays fairly good chess, but amateur tournament standards. But visitors are always disappointed to find that this program does not "learn", in the sense that it carries no permanent change away from the games it plays. They are even more disappointed in our attempts to explain why this does not disturb us very much. We claim that there is indeed an important kind of learning within the program; this is in the position-description summaries that are constructed and used as it analyzes the positions it is playing. But because board positions do not often repeat exactly in subsequent games (except for opening positions and end-games) and because the kinds of descriptions the program now uses do not have good qualities for dealing with broader classes of positions, there would be no point in keeping such records permanently.

We do not yet understand how to make the higher-level strategy-oriented descriptions that would make sense in the context of learning to improve. When we, ourselves, learn how to construct the right kind of descriptions, then we can make programs

construct and remember them, too, and the problem of "learning" will vanish. In the past, our laboratory avoided experiments with learning systems that seemed theoretically unsound, although we did not avoid studying them theoretically. This was because we believed that learning itself was not the real problem; what was needed was more knowledge about the intelligent shaping of description-handling processes. For the same reasons we avoided linguistic exercises such as Mechanical Translation, in favor of studying systems that could deal with limited fragments of meaning, and we avoided "creative" systems based on uninterpreted stochastic processes in favor of analyzing the interactions of design goals and constraints. Now we think we know enough to begin such experiments.

In the rest of this chapter we will discuss some systems that do exhibit some non-trivial learning functions. It should be understood from the start that these are not to be thought of as "self-organizing systems". They are equipped with very substantial initial structures; -- they are provided with many built-in "innate ideas".

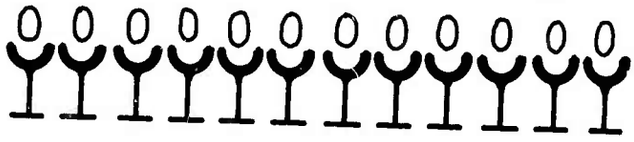
Because of this, some readers might object that although these programs learn, they do not significantly "learn to learn". Is this a serious objection? We do not think so, but the question is really one of degree and we are still much too uncertain about it to take a decisive position. In one view learning to learn would be an extremely advanced problem compared to what we now understand. In another view,it is just one more problem about certain kinds of program-writing processes, not strikingly different from the static structural situations we already understand rather well. Our position is intermediate between these, at present.

We think that learning to learn is very much like debugging complex computer programs. To be good at it requires one to know a lot about describing processes and manipulating such descriptions. Unfortunately, work in Artificial Intelligence has not, up to now, been pointed very much in that direction, so today we have little real knowledge about such matters.

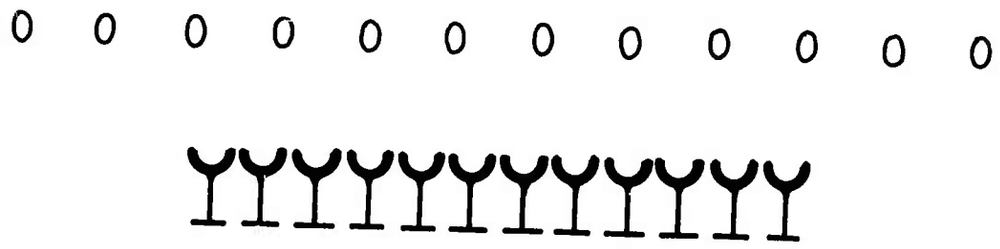
Consequently, we are in a poor position to estimate how complex must be the initial endowment of intelligent learners -- ones that could develop as rapidly as human minds rather than requiring evolutionary epochs. We certainly cannot assume from what we know that the "innate structure" requires must be very, very complex as compared to present programs. It might be much simpler. Even in the case of humans we have no useful guidelines. There is probably enough potential genetic structure to supply large innate behavioral programs but no one really knows much about this, either, at present. So let us proceed, instead, to discuss our present understanding. We begin with some experiments on natural intelligence.

4.1 An example of Learning: Piaget's Conservation Experiments

A classical experiment of Jean Piaget shows remarkably repeatable patterns of response of children (in the age range of 4-7 years) to questions about this sort of material:



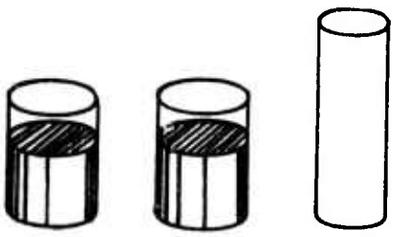
Question: "Are there more eggs or more egg-cups?"
Typical Answer: "No, the same."



Question: "Are there more eggs or more egg-cups?"
Typical Five Year Old's Answer: "More eggs."
Typical Seven Year Old's Answer: "Of course not!"

Furthering questioning makes it perfectly clear that the younger child's comparison is based on the greater "spread" or space occupied by the eggs. The older child ignores or rejects this aspect of the situation and is carried along by the "conservationist" argument: before we spread them out there were the same number of eggs and egg-cups; we neither added or subtracted any, so the number must still be the same.

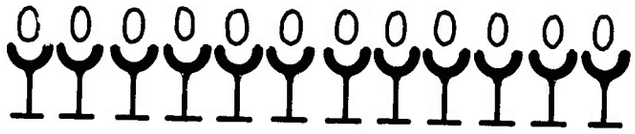
Before constructing a theory of this we describe some other situations that are similar; nothing is more dangerous than to base a theory on just one example and we want the reader to have enough material to participate and, amongst other things, make rival theories. Here is another relatively repeatable experiment. One shows the child three jars.



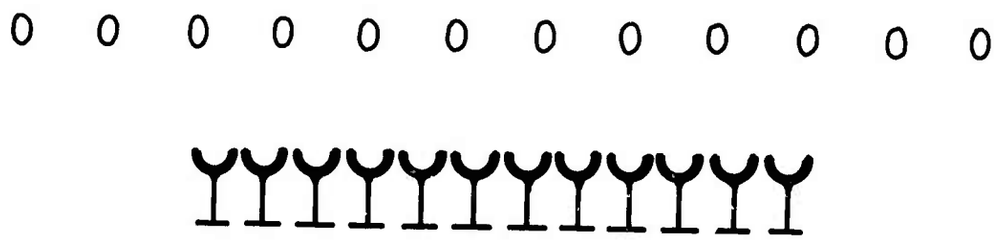
He agrees that the first two contain the same amount of liquid. Then, before his eyes, we pour the second jar into the third and ask again about the amounts. Usually, the younger child will say that the tall jar contains more; the older child says "Of course they have the same amount. It is the same water so it could not have changed."

4.1 An example of Learning: Piaget's Conservation Experiments

A classical experiment of Jean Piaget shows remarkably repeatable patterns of response of children (in the age range of 4-7 years) to questions about this sort of material:



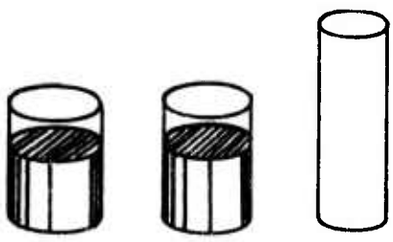
Question: "Are there more eggs or more egg-cups?"
Typical Answer: "No, the same."



Question: "Are there more eggs or more egg-cups?"
Typical Five Year Old's Answer: "More eggs."
Typical Seven Year Old's Answer: "Of course not!"

Furthering questioning makes it perfectly clear that the younger child's comparison is based on the greater "spread" or space occupied by the eggs. The older child ignores or rejects this aspect of the situation and is carried along by the "conservationist" argument: before we spread them out there were the same number of eggs and egg-cups; we neither added or subtracted any, so the number must still be the same.

Before constructing a theory of this we describe some other situations that are similar; nothing is more dangerous than to base a theory on just one example and we want the reader to have enough material to participate and, amongst other things, make rival theories. Here is another relatively repeatable experiment. One shows the child three jars.



He agrees that the first two contain the same amount of liquid. Then, before his eyes, we pour the second jar into the third and ask again about the amounts. Usually, the younger child will say that the tall jar contains more; the older child says "Of course they have the same amount. It is the same water so it could not have changed."

construct and remember them, too, and the problem of "learning" will vanish. In the past, our laboratory avoided experiments with learning systems that seemed theoretically unsound, although we did not avoid studying them theoretically. This was because we believed that learning itself was not the real problem; what was needed was more knowledge about the intelligent shaping of description-handling processes. For the same reasons we avoided linguistic exercises such as Mechanical Translation, in favor of studying systems that could deal with limited fragments of meaning, and we avoided "creative" svstems based on uninterpreted stochastic processes in favor of analyzing the interactions of design goals and constraints. Now we think we know enough to begin such experiments.

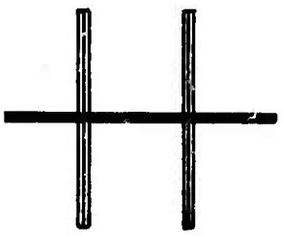
In the rest of this chapter we will discuss some systems that do exhibit some non-trivial learning functions. It should be understood from the start that these are not to be thought of as "self-organizing systems". They are equipped with very substantial initial structures; -- they are provided with many built-in "innate ideas".

Because of this, some readers might object that although these programs learn, they do not significantly "learn to learn". Is this a serious objection? We do not think so, but the question is really one of degree and we are still much too uncertain about it to take a decisive position. In one view learning to learn would be an extremely advanced problem compared to what we now understand. In another view,it is just one more problem about certain kinds of program-writing processes, not strikingly different from the static structural situations we already understand rather well. Our position is intermediate between these, at present.

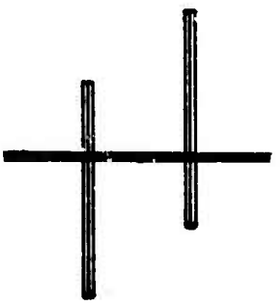
We think that learning to learn is very much like debugging complex computer programs. To be good at it requires one to know a lot about describing processes and manipulating such descriptions. Unfortunately, work in Artificial Intelligence has not, up to now, been pointed very much in that direction, so today we have little real knowledge about such matters.

Consequently, we are in a poor position to estimate how complex must be the initial endowment of intelligent learners -- ones that could develop as rapidly as human minds rather than requiring evolutionary epochs. We certainly cannot assume from what we know that the "innate structure" requires must be very, very complex as compared to present programs. It might be much simpler. Even in the case of humans we have no useful guidelines. There is probably enough potential genetic structure to supply large innate behavioral programs but no one really knows much about this, either, at present. So let us proceed, instead, to discuss our present understanding. We begin with some experiments on natural intelligence.

If we perform the pouring behind a screen, telling him what we are doing without his seeing it, the younger child also may say the amounts are the same, but may change his mind when he sees it.



In this experiment, younger children agree the rods are equal at first, but when displaced as shown at the right, the "upper" one is usually said to be longer.



How can we explain the difference between the less and more mature children. We see two problems here from the point of view of learning. First, how is the pre-conservationist view acquired (and executed); then how is it replaced by a conservationist one? To many psychologists only the second seems interesting. This is because it is tempting to explain the earlier response in terms like "the child is carried away by appearances," or"the child is dominated by its perception," that is, instead of logic. The usual interpretation, then, is that the transition requires the development of some sort of reasoning capacity that allows it to "ignore the appearance" in favor of reasoning about "the thing itself".

There are serious problems with this view, we feel. First, the "appearance" theory is too incomplete; the notion of appearance is not structured enough. Second, we know that much younger children are quite secure (in other circumstances) about the properties of "permanent objects"; they are sufficiently surprised by magic that there is no reason to suppose they lack the required "logic". We do not think they lack any really basic or primitive intellectual ingredients; rather,they lack some particular kinds of knowledge and/or procedures that are appropriate here. Our view is most easily explained by proposing a more detailed mini-theory for the performance of the non-conservation child.

Behind the "appearance" theory lies some sort of assumption that the water in the tall jar, the upper one of the rods, and the spread-out eggs appear to be "more" than their counterparts, because of some basic law of perception. We think things are more complicated than that, and postulate that the younger child when asked to make a quantitative comparison, choose to describe the things being compared in terms of "how far they reach, preferably upwards or in some other direction if necessary". That this description comes from a choice is clear from the fact that he can realiably tell which is "wider" or "taller", when it is not a question of which is "more". Indeed, if we asked the younger child to describe the situation in detail before asking which has more, he might say something like this:

(A) "There is a tall, thin column of water in the tall, thin jar and a short, wide column in the short, wide jar."

Actually, a four year old will not say anything of the sort. His syntactic structure will not be so elaborate, but more important, he is unlikely to produce that many descriptive elements in any one description. If we ask him "what is this", he might say any of "high glass", "almost full", "high water", "round", etc., depending on what he imagines at the moment as a purpose for the question or the object. In any case, if we ask him for a description after telling him we want to know which has more, he will probably say the equivalent of:

(B) "There is a high column of water in the tall jar and a low column of water in the short jar."

To answer the question "which has more" one has to apply some process to the description of the situation. Once we have the second description (B) almost any process would choose the "high column of water". We still need a theory of what symbolic rules delete preferentially the horizontal descriptive elements from the first description (A).

Another possibility is that perhaps the child is misinterpreting "more"; if he were strongly "motivated" by being thirsty or hungry he might give better answers. The experiments are, however, always careful about this, and one gets similar results if the eggs are replaced by candy actually to be eaten, or the water by a delicious beverage.

In suggesting that the child converts description "A" to description "B" we are proposing an analogy with analogy! Is this too neat? Are we inventing this process for the child, who does not really do anything so simple? Certainly, we are making a mini-theory much simpler than what really happens. But what really happens is, we believe, correspondingly simpler than what most observers of children imagine is happening! The following kind of dialog is typical of what goes on in another situation that Piaget and his colleagues have studied, and illustrates explicitly the same striking kind of transformation of descriptions:

INTERVIEWER: How many animals are there?
CHILD: Five. Three horses and Two cows.
INTERVIEWER: Are there more horses or more animals?
CHILD: More horses. Three horses and two animals.

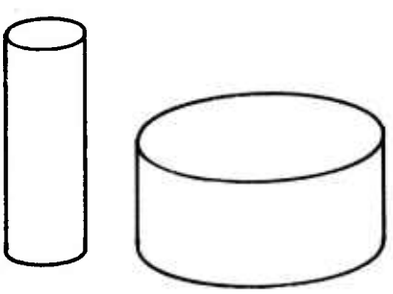
I: Now listen carefully:
ARE THERE MORE HORSES OR MORE ANIMALS?

I: What did I ask you?
C: Are there more horses or more animals?
I: What is the answer?
C: More horses.
I: What was the question again?
C: Are there more horses or more cows?

We explain this phenomenon on a similar basis; again the child has to make a comparison of quantity. He has learned that it is generally correct to do this by counting mutually exclusive classes and the worst thing is to count anything more than once. So he proceeds to describe the situation "correctly" for such purposes, and (in this frame) gets the correct answer.

It is often said that the pre-conservation child gets the answer wrong to "inclusion" questions. No. He gets the answer right. He gets the question wrong! Inclusion comparisons are never natural, so we can agree with the child that these are silly "trick" questions, anyway.

Returning to judging "amount" by height alone, we must ask what "learning" process could cause a child to acquire this "false" idea? Our mini-theory begins not by trying to explain the particular fact (why the child says this about water or that about eggs) but to look for a general rule for comparing quantities that combines simplicity with widespread utility. Who is bigger; the child or his cousin? Stand back to back! How do you divide a bottle of coke between two glasses? By the level -- and generally this is fine because the glasses are identical. Finally, the child can afford to be wrong some of the time; this rule serves very well for many purposes and it would be hard to find a better one without taking a giant step.

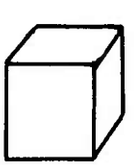


A confirmation of this is given by the children who judge that the thinner container of this pair could hold more water.

Although fewer children will say this, the fact that there are any who do disproves the "appearance" theory, for one can hardly maintain that an unalterable law of perception is operating here.

Clearly the (heuristic) symbolic rule of vertical extent here overrides "perception" of dimensions.

One could make a case for the "appearance" theory, in the water-jar experiment as follows: The water is much higher where it is high, but only somewhat wider where it is wide. The most plausible kind of comparison algorithm would look first for a unique term or quality upon which to base its decision -- as is easily found in (B). If there is none -- as in (A) -- then a subprocess has to make a "quantitative" comparison. But even this seems less symbolic than quantitative, for if we compare "much higher" with "somewhat thinner", the former will surely win! In



any case, even adults can hardly believe that these two solids could have the same volume. So, if the child were really faced with the problem of comparing quantitative dimensions, this would be almost impossible for him.

We next have to ask, how was this rule acquired, and how can we explain the transition to conservationist thinking? The simplest theory would assert that the child specifically learns each conservation (and, earlier, each comparison technique) as isolated pieces of knowledge. However, this theory is incomplete because it postulates some agent or specific circumstance responsible for the specific act of learning. A more satisfactory kind of theory would let the child himself play the part of the "teaching agent" in the weak theory, and find his own strategies for making descriptions adequate for his problems.

Consider again the original conservation-of-number experiment. Suppose that we wanted to TELL the child how to behave. An authoritarian approach would shout at him: no, no, no, they are equal. But most teachers would prefer the gentler approach of explaining what he is doing wrong. One could say: "Yes, you are right, the eggs take up more space than the egg-cups so you could say that SPATIALLY there are more eggs; but NUMERICALLY there are still as many eggs as egg-cups."

We hope readers are objecting that no child of five will understand this little speech. Indeed,one can go a step further and say that the attempted lesson begs the entire question. The non-conservation child seems to lack a sharp distinction between "numerical" and "spatial". That's his problem! If he knew how to use the distinction well enough he would not need us to teach him about conservation. Our child has already a variety of concepts about quantities; we maintain that his problem is in knowing which to use when (instead of, or combined with others) in describing situations. His real problem is that he does not yet know good enough ways to describe his descriptors! If he learned how to describe his descriptors -- for example, to label some as "spatial" and some as "numerical" -- and if he could use these descriptions of descriptors to choose the appropriate ones, then the specific problem of learning conservations would dissolve away. As it should! For "conservation" is not a single thing, and "it's development is typically spread out over several years as a child learns to deal with number, mass, volume, and other descriptive concepts.

Assuming a structure for classifying descriptions we can imagine an internal scenario, for the egg experiment, in which many descriptions are considered by a supervising process:

(1) Choose a kind of rule. Choices are
QUANTITATIVE RULES
HISTORICAL RULES
.
.

(2) QUANTATIVE is chosen. Select a kind. Choices are
SPATIAL
NUMERICAL

.
.

(3) SPATIAL is chosen. Select a kind. Choices are
EXTENT implies more
SPARSENESS implies less
.
.

(4) Try EXTENT. The spread out eggs have more extent.
This means MORE.

(5) Test for coherence with other SPATIAL rules? Try SPARSENESS. The eggs are sparser.
This means LESS!

An inconsistency. Reject or explain.
Reject method

(3') Try NUMERICAL.
Try COUNTING
Too many to count.
Reject method

(2') Reject choice of quantitative rules!
Try the next choice, HISTORICAL
.
.

When HISTORICAL is tried, one might first choose IDENTITY. Some eggs were moved, but none added or taken away
This means SAME!
Test for coherence with other HISTORICAL rules.
Try REVERSIBILITY. The operation SPREADING-OUT is reversible.
This means SAME!

We conclude that HISTORICAL seems consistent.

The same sort of scenario could be constructed for the water experiment; there the counting descriptions cannot be invoked, but instead other quantitative descriptions must be available. In each attempt, the description of the scene takes on a different form: the successful historical form will resemble

> "The water that was in the second jar is now in the third jar"

and "of course" it has the same amount as the first jar! Well! This gives the right answer, because he has obtained an adequate description. What kinds of processes must he have in order to do this. We have already proposed that he has a procedure for selecting descriptions; in what kind of environment could this operate? One kind of model would assume that the mature child's description is at first more elaborate, including both geometric and historical elements,

> "The amounts of water in the first and second jars were equal. The water that was in the second jar is now in the third jar. The water in the third jar is higher and thinner than that in the first jar."

The mature child, we might theorize, will eliminate elements

from his description until there are no serious conflicts. This will yield a tentative answer, which he can maintain if he can explain away any problems that arise from reconsidering other details. Alternatively, one might imagine a process that begins with a very primitive description and elaborates it. But in any case, the process must have facilities for such functions as:

> Choosing among the most plausible methods for answering the question. To apply a method he must bring the description into a useable form. For example, when he chooses a "history" method he suppresses some features of the spatial appearance. This means he must have a good classification of the different kinds of description elements.
>
> The selection of the description involves common-sense knowledge. This, in a word, means that his entire cognitive structure is potentially engaged -- language, goals, logic, even interpersonal situational processes.
>
> If the situation is at all novel,then any committment to "ignore" a class of elements may require a reason or "excuse", for conflicts in the original description that remain unexplained. A standard strategy is "compensation" -- knowing when it is reasonable to propose tradeoff between such pairs as height and width when manipulating fluids.
>
> One cannot balance an arbitrary pair of dimensions, and particular pairs compensate only under suitable conditions. Ideas like "geometric property" are necessary, so that one isn't tempted to trade height with color, for example. What features of histories might correspond to such static properties as "spatial" and "numerical"?
>
> Most important, the directing process in which the history of the situation wins out over the unusable geometric features,must exist and be debugged well enough that is can be relied upon! The child needs to have and trust the higher-order knowledge about which kinds of knowledge should have priority in each situation.

We have intentionally not specified the time scale of this scenario; some of it occurs over long periods, while some in the course of solving a particular problem. Furthermore, these conditions are still incomplete, yet our structure is already quite complicated. But so is the situation! Remember, our child can already carry on an intelligent conversation. This is not a good place to encourage the use of Occam's Razor. The time for that is when one has several good competing theories, not before one has any! It takes the child several years to work out all of this, and a theory that explained it away on too simple a basis might be therefore suspect.

We do not, we repeat, want to explain the different conservations either on completely separate bases or by one unifying principle. We want to see it as the outcome of an improvement in the child's procedures for dealing with the variety of descriptions that he comes into possession of.

In the traditional "theories of learning" there was a tendency to ask

> "How does such-and-such a "response become connected to such-and-such a "stimulus".

We now see that the proper questions are much more like

> "How can such-and-such a procedure be added to the descriptive or deductive systems"

4.2 Learning

A serious complaint about the heuristic programs of the past was their very limited ability to learn. This made them too inflexible to be useful except in very special situations. Over the years many direct attempts to construct "learning programs" led to very indifferent results. There is a close analogy, we feel, between this and the similar situation in the history of constructing psychological theories of learning.

If a child were to learn that 7+5=12 and 39+54=93 and, say, one hundred other such "responses", we would not agree he had learned to add. What is required is that he learn an appropriate procedure and how to apply it to numbers he has never used before. Another side of this "stimulus-response" problem: just as in the Analogy situation, the secret of learning often lies in the discovery of descriptions that emphasize the "essential" aspects of things or events, and omit or subjugate the "accidental" features. It would do us little good to remember that some particular thing happened in exactly a certain situation, since identical conditions never recur.

We do not need,or want, to remember the precise details of a broken chair, but we do want to remember that bad things happen when chairs have broken rungs -- for that is an essential difference between this and a usable chair. Indeed, the greater our knowledge and powers of observation, the more selective must be our choice of descriptions, because of the magnified problem of becoming lost in searching through networks of irrelevant details.

Finally, one hears complaints of the form "You programmed it to do that! It didn't learn it by itself!". There is a spectrum of degrees of autonomy in learning activities, and one wonders what are the distinctive features of importance between a child learning while playing by himself, discovering things under the shrewd guidance of an attentive instructor, prying a theory out of a mediocre textbook, and having it explained directly and concisely by a superb expositor.

It is tempting to try to disentangle this messy web of different phenomena. The appearance of an impossibly refractory problem in science is often the result of fusing fundamentally different problems (each of which may be relatively simple) when there is no common solution to the whole set. We think this is true of the many different ways in which programs can be said to learn. But despite this diversity there are important common themes. Most important of these, we feel, is the need for enough descriptive structure to represent the relation between learning situations and the concepts learned from them. Another theme comes from noticing that the kinds of learning we have found most difficult to simulate are those that involve a large stock of prior knowledge and analytic abilities. This leads us to propose for study very pure forms of the problem of handling diverse kinds of knowledge -- prior to worrying about the problems of acquiring such knowledge. To separate out these strands we will consider at various points such not-entirely-separable ideas of "learning" as these:

Learning by development or maturation
Learning without description (by quantitative adaptation)
Learning by building and modifying description
Learning by being taught
Learning by Analogy
Learning by being told
Learning by being programmed
Learning by understanding

4.3 Learning Without Desciption -- "Incremental Adaptation"

There is a large literature concerned with clustering methods, scaling, factor analysis, and optimal decision theories, in which one finds proposals for programs that "learn" by successive modifications of numerical parameters. An outstanding example of this is seen in one of the well-known programs of A Samuel, that plays a good game of Checkers. Other examples abound; all perceptron-like "adaptive" machines, all "hill-climbing" optimization programs, most "stochastic learning" models using reinforcement. Some details can be found in the later chapters of our book, PERCEPTRONS.

The conclusions drawn in PERCEPTRONS are too technical to review here in detail, but we can describe the general picture that emerges. Within the classes of concepts that these machines can represent, that is, describe as rather literal "sums" of already programmed "parts" -- the learning abilities are effective and interesting. However, the descriptive powers of these quasi-linear learning schemes have such peculiar

and crippling limitations that they can be used only in special ways. For example, we can construct, by special methods, a perceptron that could learn either to recognize squares, or to recognize circles. But the same machine would probably not be able to learn the class of "circles or squares"! It certainly could not describe (hence learn to recognize) a relational compound like "a circle inside a square".

These limitations are very confining. It is true that such methods can be useful in "decision-making" and diagnostic situations where things are understood so poorly that a "weighted decision" is better than nothing! But we think it might be useful to put this in perspective by assigning it as an example of a new concept of TERMINAL LEARNING. The basic problem with this kind of "learning program" is that once the program has been run, we end up only with numerical values of some parameters. The information in such an array of numbers is so homogeneous and unstructured -- the "weight" of each "factor" depends so much on what other factors are also involved in the process -- that each number itself has no separate meaning. We are convinced that the results of experience, to be useful to "higher level processes", must be summarized in forms that are convertible to structures that have at least some of the characteristics of computer programs -- that is, something like fragments of program or descriptions of ways to modify programs. Without such capabilities, the simple "adaptive" systems can "learn" some things, to be sure, but they cannot learn to learn better! They are confined to sharpening whatever "linear separation" or similar hypotheses they are initially set to evaluate. A terminal learning scheme can often be useful at the final stage of a performance or an application, but it is potentially crippling to use it within a system that may be expected later to develop further.

One could make similar criticisms of another aspect of the adaptive "branch and bound" procedures found in most game-playing and other heuristic programs that follow the "look-ahead and minimax" tradition. Suppose that in analyzing a chess position we discovered that the KB-2 square is vulnerable to a rook-queen fork by moving a knight to that square. The traditional program returns a low numerical value for that position. What it really should do is return a description of why the position is bad. Then the previous plausible-move generator can be given a constructive suggestion: look for moves that add a defense to that square,or threaten one of the attacking pieces, etc. Subsequent exploration will discover more such suggestions. Eventually, these conditions may come to conflict logically, e.g., by requiring a piece to attack two squares that cannot both lie in its range. At this point, a deductive program could see that it is necessary to think back to an earlier position. Similarly, a description of that situation, in turn, could be carried further back, so that eventually the move generator can come to work with a knowledgeable analysis of the strategic problem. Surely this is the sort of thing good players must do, but no programs yet do anything much like it.

This argument, if translated into technical specification, would

say that if a chess program is to "really" analyze positions it must first have descriptive methods to modify or "update" its state of knowledge. Then it needs ways to "understand" this knowledge in the sense of being able to make inferences or deductions that help decide what experiments next to try. Here again, we encounter the problem of "common sense" knowledge since,although some of this structure will be specific to chess, much also belongs to more general principles of strategy and planning.

People working on these homogeneous "adaptive learning" schemas (either in heuristic programming or in psychology) are not unaware of this kind of problem. Unfortunately, most approaches to it take the form of attempting to generalize the coefficient-optimizing schema directly to multi-level structures of the same kind, such as n-layer perceptrons. In doing so, one immediately runs into mathematical problems: no one has found suitably attractive generalizations (for n levels) of the kinds of convergence theorems that, at the first level, make perceptrons (for example) seem so tempting. We are inclined to suspect that this difficulty is fundamental -- that there simply do not exist algorithms for finding solutions in such spaces that operate by successive local approximations. Unfortunately we do not know how to prove anything about this or, for that matter, to formulate it in a respectably technical manner.

We could make similar remarks about most of the traditional "theories of learning" studied in Psychology courses. Almost all of these are involved with the equivalent of setting up connections with the equivalent of numerical coefficients between "nodes" all of the same general character. Some of these models have a limited capacity to form "chains of responses", or to cause some classes of events to acquire some control over the establishment of other kinds of connections. But none of these theories, from Pavlov on, seem to have adequate ability to build up processes that can alter in interesting ways the manner in which other kinds of data are handled. These theories are therefore so inadequate, from a modern computation-theory view, that today we find it difficult to discuss them seriously.

Trial and Error

Why, then,have such theories been so persistently pursued? The followers were certainly not naive about these difficulties. One influence, we think, has been a pervasive misconception about the role of multiple trials, and of "practice", in learning. The supposition that repeated experiences are necessary for permanent learning certainly tempts one to look for "quantitative" models in which each experience has a small but cumulative effect on some quantity, say, "strength-of-connection".

> In the so-called "stimulus-sampling" theories we do see an attempt to show how certain kinds of one-trail learning processes could yield an external appearance of slow improvement. In this kind of theory, a response can become connected with many different combinations of stimulus features or elements as a result of a

> sampling processes. In each learning event a new combination can be tried and tested. This is certainly closer to the direction we are pointing. However, we are less interested in why it takes so many trials to train an animal to perform a simple sequence of acts, and more interested in why a child can learn what a word means (in many instances) with only a single never-repeated explanation.

What is the basis for the multiple-trial belief? When a person is "memorizing" something he may repeat it over and over. When he practices a piece of music he plays it over and over. When we want him to learn to add we give him thousands of "exercises" When he learns tennis he hits thousands of balls.

Consider two extreme views of this. In the NUMERICAL theory he moves,in each trail, a little way toward the goal, strengthening the desired and weakening the undesired components of the behavior. In the SYMBOLIC view, in each trial there is a qualitative change in the structure of the activity -- in its program. Many small changes are involved in debugging a new program, especially if one is not good at debugging! It is not a matter of strengthening components already weakly present so much as proposing and testing new ones.

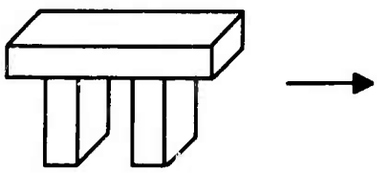
The external appearance of slow improvement, in the SYMBOLIC view, is an illusion due to our lack of discernment. Even practicing scales, we would conjecture, involves distinct changes in one's strategies or plans for linking the many motor acts to already existing sequential process-schema in different ways, or altering the internal structures of those schemas. The improvement comes from definite, albeit many, moments of conscious or unconscious analysis, conjecture, and structural experiment. "Thoughtless" trials are essentially wasted.

To be sure, this is an extreme view. There are, no doubt, physiological aspects of motor and other learning which really do require some repetition and/or persistence for reliable performance. Our point is that the extent of this is really quite unknown and one should not make it the main focus of theory-making, because that path may never lead to insight into the important structural aspects of the problem. In motor-skill learning, for example, it is quite possible one needs much less practice than is popularly supposed. It takes a child perhaps fifteen minutes to learn to walk on stilts. But if you tell him to be sure to keep pulling them up, it takes only five minutes. Could we develop new linguistic skills so that we could explain the whole thing? We might conjecture that the "natural athlete" has no magical, global, coordination faculty but only (or should we say "only"!) has worked out for himself an unusually expressive abstract scheme for manipulating representations of physical activities.

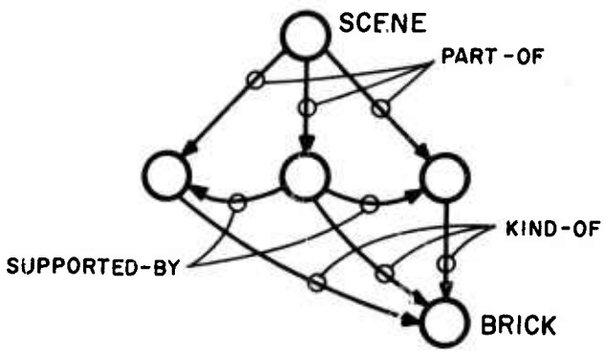
4.4 Learning by Building Descriptions

We can illustrate much more powerful concepts of learning in the context of a procedure developed by P. Winston to learn to

recognize simple kinds of structures from examples. Like the SEE program of Guzman (which it uses as a sub-process) it works in the environment of childrens' building blocks. When presented with a scene, it first observes relations between features and regions, then groups these to find proposed structures and objects, and then attempts to identify them (using description-matching methods and the results of earlier learning experiences). Thus, the simple scene on the left is described by a network of abstract objects, relations, and relations between relations.

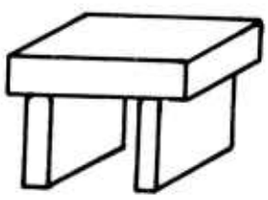


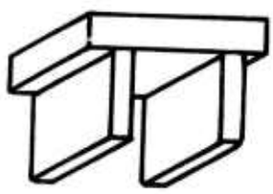
SCENE 1 : AN ARCH



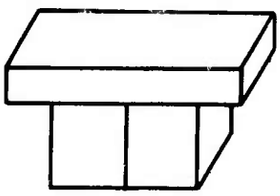
In this diagram, the heavy circles represent particular physical objects, the other circles represent other kinds of concepts, and the labels on the arrows represent relations. The program is equipped from the start to recognize certain spatial relations such as contact, support, and some other properties of relative position. We tell the machine that this is (an example of) an ARCH, and it stores the description-network away under that title.

Note that since these properties describe only relative spatial relations, the very same network serves to describe both of these figures, which are visually quite different but geometrically the same.

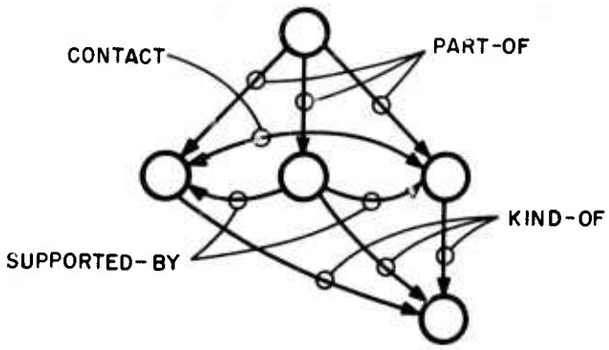




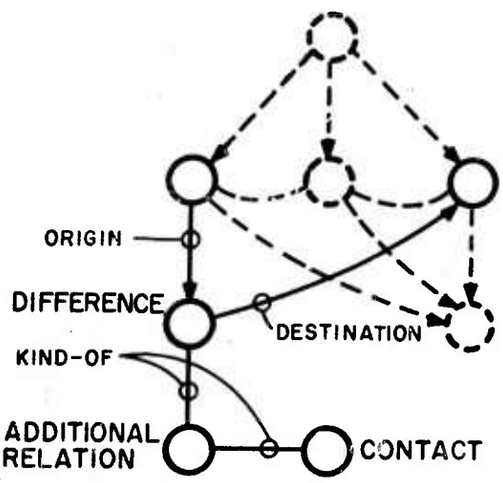
Next we present SCENE 3, to the left below, and the machine constructs the network shown to its right.



SCENE 2 : NOT AN ARCH

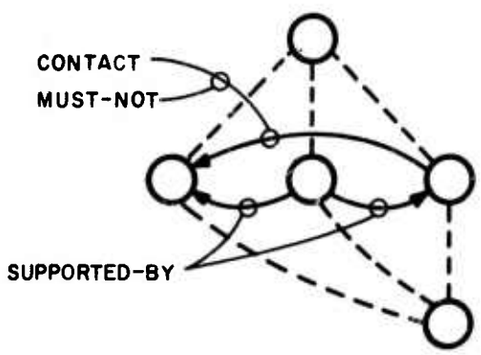


This differs from the network of SCENE 1 in only a few respects. If the program is asked what this structure "is", it will compare this description with others stored in its memory.

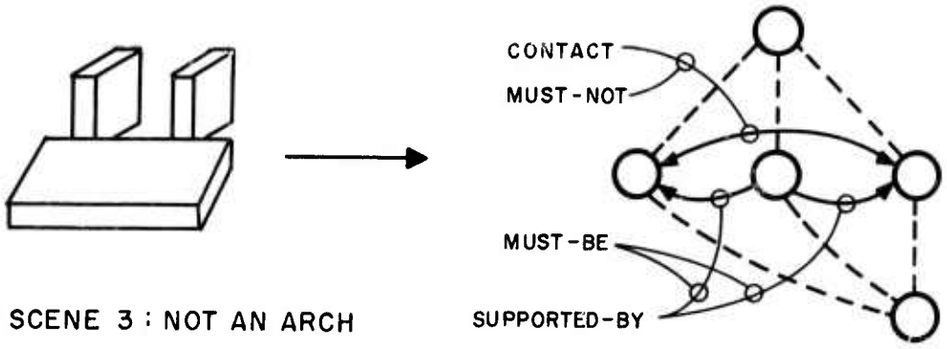


It has already networks for tables, towers, and a few other structures but, as one might expect, the structure it finds most similar is the ARCH description stored just a moment ago. So it tentatively identifies this as an arch. In doing this, it also builds a descriptive network that describes the difference between scene 1 and scene 2, and the difference is represented somewhat like this.

Now we tell the machine that scene 2 is NOT an example of an ARCH. It must therefore modify its description of "ARCH" so that structure 2 will no longer match the description, hence will no longer be "seen" as an ARCH. The method is to add a "rejection pointer" for the contact relation.

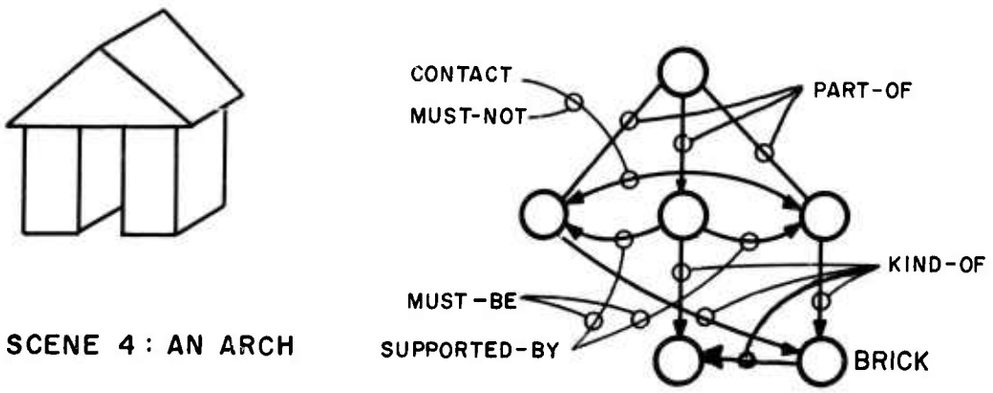


Now for the next example: we present scene 3 and assert that this, too, is not a ARCH. The most prominent difference, in this case, is that the new structure lacks the support relations



SCENE 3 : NOT AN ARCH

and the program for modifying "ARCH" now adds an "enforcement pointer" to the support relations. Finally, we present another example, scene 4, and assert that this is an acceptable example of an ARCH.



SCENE 4 : AN ARCH

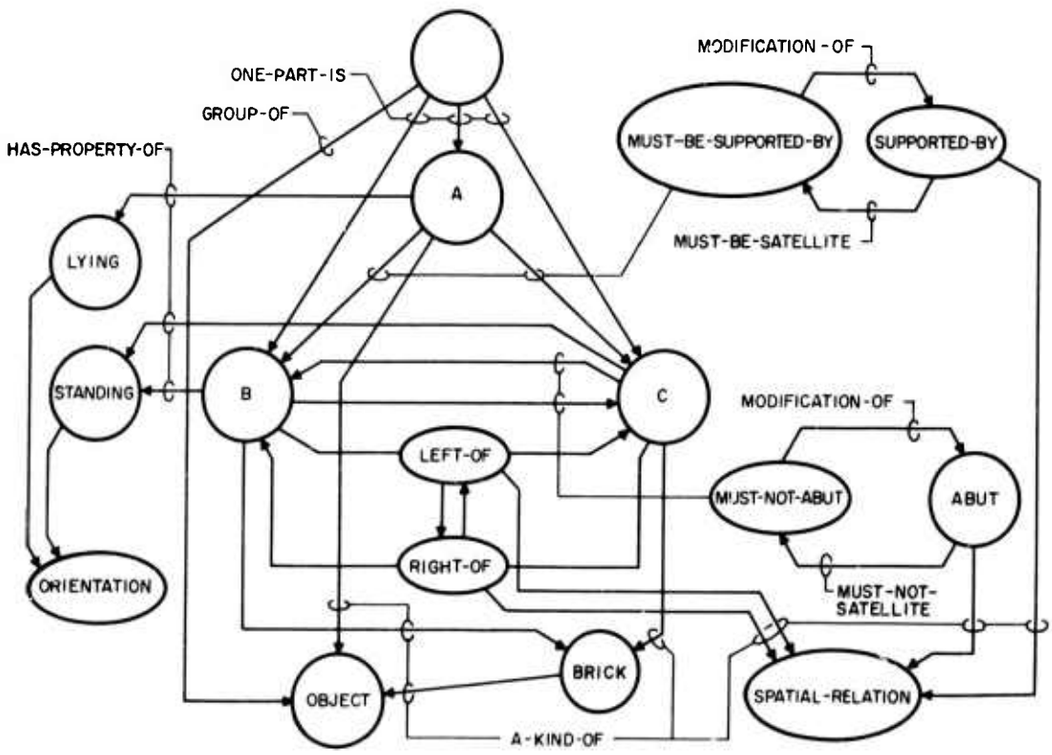
The most important difference, now, is the shape of the top block. The machine has to modify the description of "ARCH" so that the top block can be either a brick or a wedge. One strategy for this would be simply to invent a new class of objects -- "brick-or-wedge". This would be extremely "conservative", as a generalization or explanation. Winston's strategy is to look in memory for the smallest class that contains both bricks and wedges. In the machine's present state the only existing such classes are "prism" and "object" -- the latter is the class of all bodies, and includes the "prism" category, so the new description will say that the top object is a kind of prism. If we replaced the wedge by a pyramid, and told it that this, too, is an arch, it would have to change the top object-description to "object", because this is the smallest class containing "brick" and "pyramid". Now we can summarize the program's conclusion: an arch is

> "A structure in which a prismatic body is supported by two upright blocks that do not touch one another."

We have just seen how the program learns to identify correctly the membership of scenes 1-4 as to whether they are ARCHES or not. As a consequence, it will probably "generalize" automatically to decide that

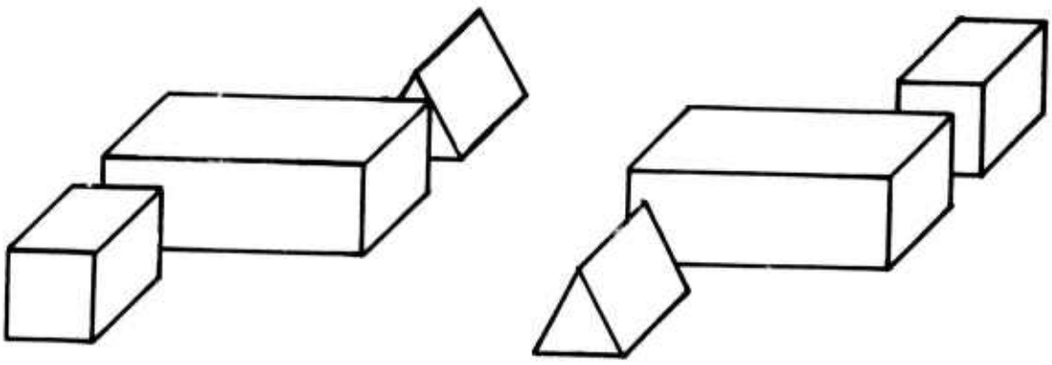
are also arches, because there are no "must-be-a..." enforcement pointers to either the supports or the top. Of course this judgement really depends on the machine's entire experience, i.e., on what concepts are already learned, and upon details of the comparison programs.

We have suppressed many interesting details of the behavior of Winston's program, especially about how it decides which differences are "most important". For example, the final form of the network for "ARCH" is more like:



than the simple schemata shown earlier.

While on the subject, it should be noticed that within the network are represented relations between relations, as well as objects, properties and simple relations. There are important advantages to this when it comes to construction of the difference-descriptions. If the comparison program can be told that the difference between "IN-FRONT-OF" and "BEHIND", as well as that between "LEFT-OF" and "RIGHT-OF", can both be described in terms of "vertical axis symmetry", then it can be programmed to observe that all (high-level) differences between the two scenes in:



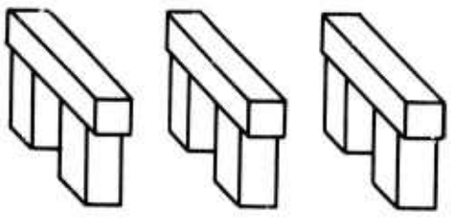
can be"explained" on this basis, hence differ only in respect to a vertical axis rotation. This is an example of a beautifully abstract form of description manipulation that, psychologically, would traditionally be attributed to something much more like an imaginary graphical rotation of the scene -- (as through there were no critically complicated problems in that reconstruction). In his thesis, Winston has only initiated such studies, and we know little about how far one can go with these methods. How much more structure would one need, to be able to learn, from examples, such concepts as symmetry? How difficult will it be to adapt such a system to learning new procedures, instead of structures? At first this might seem a huge step, but the ideas in the next section, on describing groups and repetitive structures, make the gap seem to become smaller.

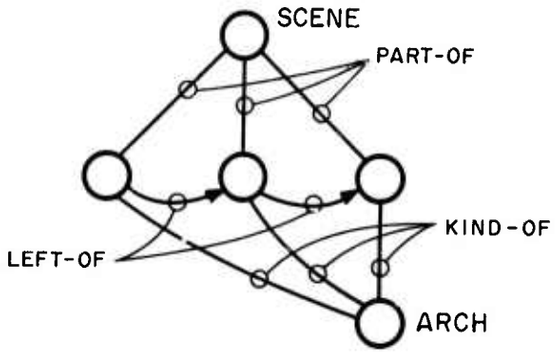
We shall see that the advantages of having a description for a "concept" (rather than just a competence) are absolutely crucial for further progress. These advantages include:

> The ability to compare and contrast descriptions (as we shall see in section 4.6)
>
> The ability to make deductions involving the concept, to adapt it to new situations.
>
> Combining several descriptions to make new concepts.

An example of the latter: Every structural "concept" that Winston's program acquires is automatically incorporated within its own internal descriptive mechanisms. Thus, if the machine were presented with the nine-block scene following, before learning a concept of ARCH, it would have produced an impossibly complex and almost useless network of relations between the

nine blocks. But after learning ARCH, it will now describe it in a much more intelligent way:





because its descriptive mechanisms proceed from local to global aggregates using as much available knowledge as it can apply. In doing this we encounter, now on a higher level, grouping problems very much like those we saw in our sketch of Guzman's SEE program, and in many cases one can adopt analogous strategies.

4.5 Learning by Being Taught

Imagine a child playing with a toy car and his blocks. He wants to build an interesting structure to play with. If the use of Winston's program were present, he could teach the child how to make an arch by the process just described, for it is not hard to convert the above description into a procedure for building arches. In fact, in Chapter 5, we shall give a sketch of exactly how this can be done! This is precisely what Winograd's program does when it translates from the semantic analysis of an object-describing noun-phrase into a robot program for building with blocks! See Chapter 5.

It is not necessary for the child to have a teacher, however. In the course of "playing" he can try experiments with the blocks and the car, and he can recognize "success" in either of these cases, among others:

a) He knows how to recognize an "ARCH" once it is built -- but does not know how to describe or to build it.

b) He has a functional play-goal: construct a road-problem for himself that is not too easy and not too hard -- such as an obstacle that requires two hands to overcome, but cannot be negotiated trivially with one hand.

In case (a) he knows how to tell which structures are in the class. In case (b), while experimenting he will indeed find that Scene 1 is good, Scene 2 is imposssible, Scene 3 is too easy, an Scene 4 (discovered as the simplest variant of the successful Scene 1) is also good. Here we get the same overall effect -- through the same mechanism -- yet in humanistic terms the behavior would be described much more naturally in terms of "exploratory", or "play", or "undirected" activity. The final result, if described in structural terms, is again

> "a structure in which an object is supported by two upright bricks that do not touch one another."

This is certainly not a perfect logical equivalent of the adult's idea of an arch; nor does it contain explicitly the idea of a surrounded passage or hole. Still, for the playing child's purposes, it would represent perhaps an important step toward formulation and acquisition of such concepts.

Again we have left alone some very important loose ends. We have concealed in the catch-all expressions "play" or "exploration" some supremely important conditions that must be fulfilled -- and at early stages of child development they won't be, and the things that are learned during "play" will be different!

> The child must already be equipped with procedures that have a decent chance of generating plausible structures.
>
> To do this, he must be able to describe to some extent why an experiment is unsatisfactory. If he cannot get his car between the supports, he must be able to think of moving the supports apart. This is not very hard, since pushing against the obstacle will sometimes do this.
>
> Since most experiments not carefully planned lead to useless structures, he has to have some ability to reconstruct a usable version of earlier and better situations after a disaster.

Without the teacher, it is unlikely that he will get good results after just four trials! He must have enough persistence in his goal-structure to carry through. To do this consistently would presuppose a good assessment of the problem's difficulty. Of course, if this is missing, he will find something else to do; not all play is productive!

Winston's program seems to be a reasonable model for kinds of behavior that would be plausible in, if not typical of, a child. The "concept" the program will develop, after seeing a sequence of examples chosen, on the order in which they are presented, and of course on the set of concepts the program has acquired previously. In many cases the experimenter may not get the result he wants; presenting examples in the wrong order could get the program (or child) irrepairably **off** the t**rack,** and he might have to back up -- or perhaps restart at an earlier stage. We cannot expect our concept-learning programs to be foolproof any more than a teacher can expect his instructional technique always to work. The teacher always risks failure until he acquires correct insights into what has happened in the student's mind.

Of course there are many small but important details of how the program decides what to do at each step, which differences to give highest priority, which parts of the description networks should be matched, what explanations it should assign to the differences that are noticed.

Thus, in building with blocks, the relations "support" and "contact" ought to dominate properties of color, particular shapes and even other spatial relations like "in front of" or "to the right of."

In a different realm of activity, a different set of priorities might be essential, lest learning be slow or simply wrong. So, one can conclude that we must also develop intermediate structures in "learning to learn" a prerequisite to a child's (or machine's) mastery of mechanical structures will be some preparation in acquiring, grouping, and interrelating the more elementary descriptive structures to be used in assembling, comparing and modifying the representations to be used in the performance-level learning itself. This is exactly the conclusion we reached, in 4.1, about the requirements implicit in "maturation".

4.6 Analogy, Again

Now we can return to our very first topic, solving problems involving analogies. In section 1.1 we proposed that the key idea would lie in finding ways to describe changes in descriptions. But this is exactly what happens in the program we have just described. When asked to describe a new scene situation, Winston's program makes use of the other descriptions it remembers, so that it can describe the scene in terms of already-learned concepts. Although we have not explained in detail how this is done, it is important to mention that the result of comparing two descriptions, in this system, is itself a description! Basically, the comparison works this way:

1. The two descriptions are "matched together", using various heuristic rules to decide which nodes probably correspond.

2. We create a new network, whose nodes are associated with pairs of nodes from the two descriptions that were matched. This is the skeleton of the comparison-description.

3. We associate with each node of this skeleton, a "comparison note" describing the correspondence. If the descriptions immediately local to two "corresponding" nodes are the same, the comparison-note is trivial. But if there are differences, (e.g., if one is a brick and the other a wedge) the "comparison note" describes this difference. Since these descriptive elements have the same format as by the original scene descriptions, one can operate upon them with the same programs. In particular, two difference-descriptions can be compared as handily as any other pair of descriptions.

Now we can apply this idea to the analogy problem. The machine must select that scene X (from a small collection of alternatives) which best completes the statement

A is to B as C is to X

That is, one must find how B relates to A and find an X that relates to C in the same way. Using the terminology

Diff[A:B]

to denote the difference-description-network resulting from comparing A with B, we simply compare the structures resulting from:

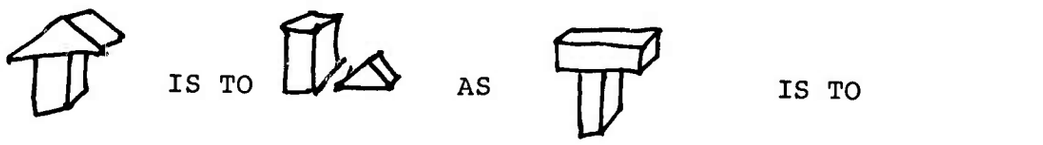
Diff[Diff[A:B] : Diff[C:X1]],

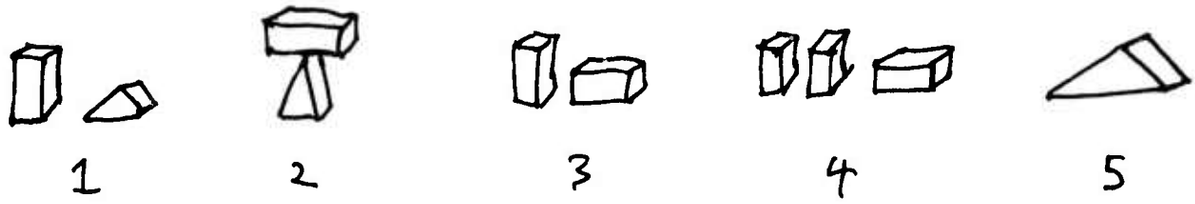
Diff[Diff[A:B] : Diff[C:X2]],

Diff[Diff[A:B] : Diff[C:X3]], etc.

Each of these summarizes the discrepancies within the "analogical explanations" for each corresponding possible answer. So to make the decision, we have to choose the "best" or "simplest" of these. We will not give details of how this is done; it is described in Chapter 7 of Winston's thesis. But note that some such device was needed already for the basic ability to identify a presented scene most closely with one of the descriptive models in memory. Thus the program must incorporate, in its comparison mechanism, conventions and priorities about such matters as whether the difference between Right and Left is to be considered simpler than the difference between Right and Above.

In this example





the machine chooses THREE as its answer, ONE as its second choice. In the slightly altered problem

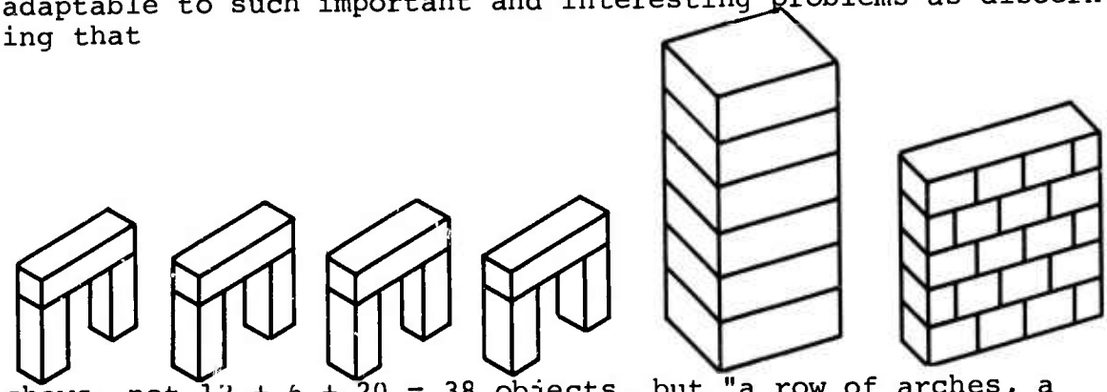
 IS TO  AS  IS TO

(same 5 figures)

It chooses FOUR as its answer.

4.7 Grouping and Induction

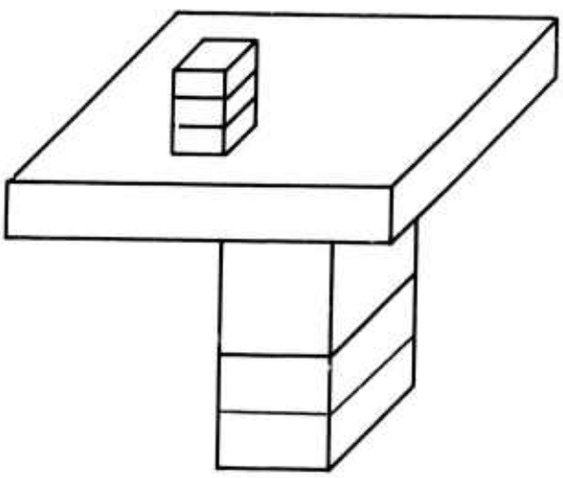
The problem of recognizing or discerning grouping or clusterings of related things is another recurrent concern not only in Psychology, but also in statistics, artificial intelligence, theory of inductive inference; indeed, of science and art in general. Most studies of "clustering" have centered around attempts to adapt numerical methods from the theory of multivariate statistics to group data into subsets that minimize some formula which compares selected inter- and intra-group measures of relatedness. But such theories are not easily adaptable to such important and interesting problems as discerning that



shows, not 12 + 6 + 20 = 38 objects, but "a row of arches, a tower of cubes, and a brick wall." More subtly, how do we "know" that one of these is three wedges while the other is three blocks? Visually, the lower objects in each tower are the same. These problems, too, can be treated by the same general methodology used in our approach to Analogy and to Learning of structures in scene-analysis.

> On many occasions we have been asked why the A.I. Laboratory is so concerned with special problems like machine vision, rather than more general approaches and problems about intelligence. In the early stages of a new science one proceeds best by gaining a very deep and thorough understanding of a few particular problems; that way one discovers important phenomena, difficulties, and insights, without which one risks fruitless periods of speculations and generalities. If the reader can see the present discussion in terms of general problems about induction and learning, the fruitfulness of the approach should speak for itself; we cannot imagine anyone believing the usefulness of these ideas is in any important way confined to description of visual or mechanical structures!

Take the groupings in the preceding firures and ask: "What qualities of the scene-descriptions characterize the intuitively acceptable groups." In some groups, like those shown above, it seems clear that the important feature is a CHAIN, say, of supported-by or in-front-of relations. In other cases it seems obvious that several objects show a common relationship to another. But no simple rules work in all situations.

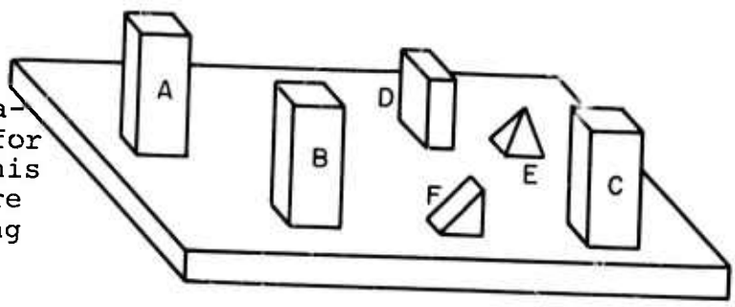


In this scene one does not usually see a single group or tower of seven blocks. Whether it is appropriate to describe this as "a seven-block stack," or as "a three-block stack supporting a plate that in turn supports a three-block stack," or as yet something else, depends on one's current purposes, orientations, or specifically on what grouping criteria are currently activated for whatever reason.

In some situations the discrepancies in the individual properties of the blocks should cause the grouping procedure to separate out the three-block stacks in spite of the fact that the support-relation chain continues through all seven blocks.

We next summarize some experiments along this line, again reporting results from P. Winston's dissertation. In Winston's grouping program, a generous hypothesis is followed by a series of criticisms and modifications.

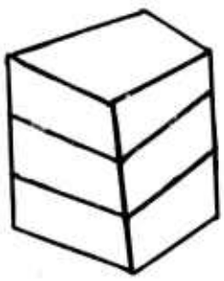
For example, when several objects have the same or very nearly the same description, they are immediately taken as candidates for a group. The blocks on this table are typical. All are polyhedra, all are standing and all are supported by the board.

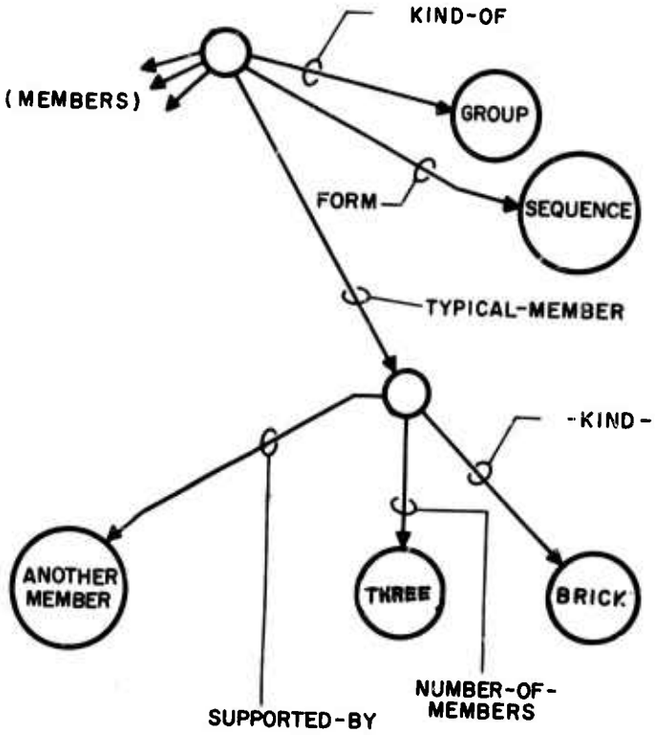


This proposal is then examined to eliminate objects which seem atypical, until a fairly homogeneous set remains. To do this, a program lists all relationships exhibited by more than half of the candidates in the set.

When the procedure operates, the first pass through the loop rejects E and F, mainly on the basis of shape. (Size is not considered in this pass because the six objects are too heterogenous for "size" to be put on the common-relationships list.) In a second pass, however, more than half the remaining objects share the "medium" size property, and block D is rejected, mainly because it does not share this property. So, finally, the procedure accepts only A B and C into the group. Obviously this is appropriate for some goals, but not others.

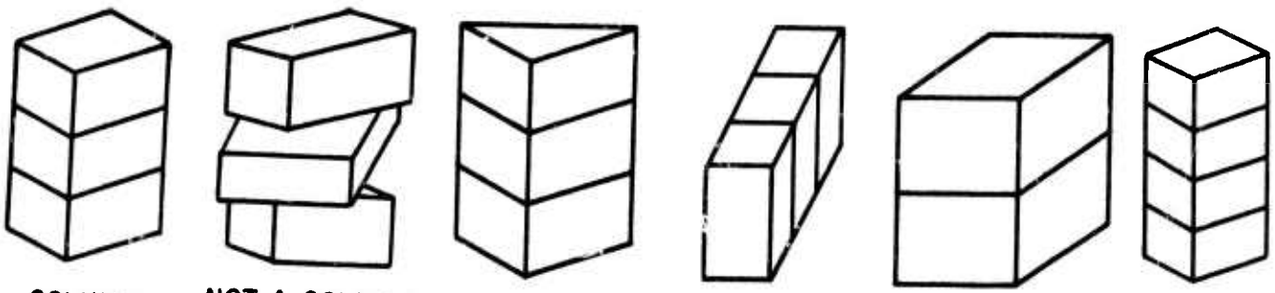
When grouping concepts are injected into the description framework, there can be unexpected and exciting consequences for other problems of induction. The figure on the next page shows the network representation obtained when the grouping process operates on the description of this 3-block column.





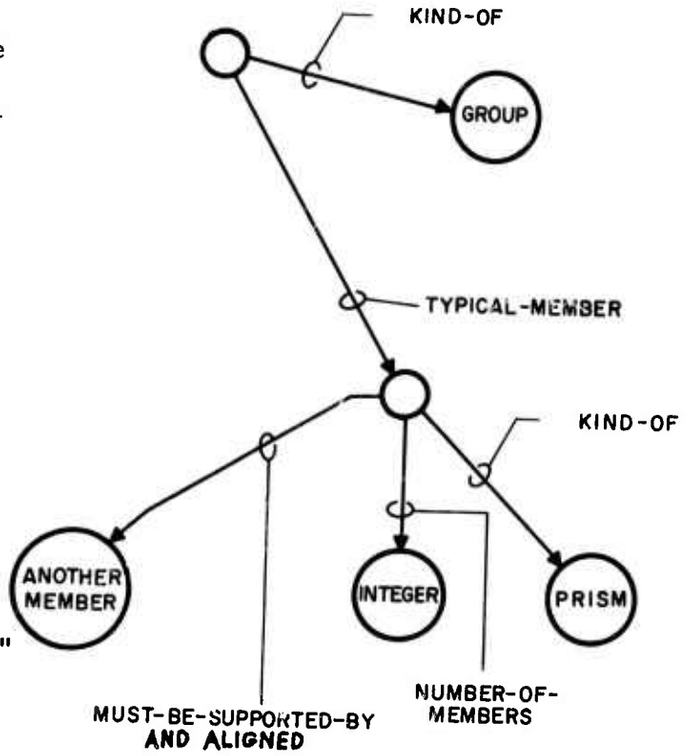
Into the description is introduced a "typical member" to which is attributed the common properties discerned by the grouping procedure. In this case, chaining was used to form the group and the description includes the fact that there were three elements in the chain.

In a learning experiment, the program is presented with the depicted sequence of scenes shown below and is told that the first, third, and sixth are instances of "column" while the others are not.



The second example causes the enforcement of a new pointer, "ALIGNED", a concept already available to the program that refers to the neat parallel alignment of edges. The third example tells the system that the typical member can be a wedge or a brick; the smallest common generalization here is "PRISM" so now a "column" can be any neatly piled stack of prisms. The fourth example changes "supported-by" to "must-be-supported-by"; the fifth, which is not seen as a group because it has only two elements, changes "one-part-is-a group" to "one-part-must-be a group".

The sixth and final example is of particular interest with respect to traditional induction questions. Comparison of it with the current concept of "column" yields a difference-description whose highest-priority feature is the occurrence of "FOUR" instead of "THREE," in the number-of-members property of the main group. What is the smallest class that contains both "THREE" and "FOUR?" In the program's present state, the only available superset is "INTEGER." Thus we obtain this description of "column" which permits a column to have any number of elements!



Is this too rash a generalization to make from so few examples? The answer depends on too many other things for the question to make much sense. If the program had already some concept of "small integer," it could call upon that. On a higher level we could imagine a program that supervised the application of any generalization about integers, and attaches an auxiliary "warning" pointer label to conclusions based on marginally weak evidence. We are still far from knowing how to design a powerful yet subtle and sensitive inductive learning program, but the schemata developed in Winston's work should take us a substantial part of the way.

Finally, we note that in describing a sequential group in terms of a typical member and its relations with the adjacent members of the chain, we have come to something not too unlike that in programming languages that use "loops," entry, and exit conditions. Again, a structure developed in the context of visual scene-analysis suggests points of contact with more widely applicable notions.

5.0 Knowledge and Generality

We now turn to another set of questions connected with our long-range goal of understanding "general intelligence". An intelligent person, even a young child, is vastly more versatile than the "toy" programs we have described. He can do many things; each program can do only one kind of thing. When one of our programs fails to do what we want, we may be able to change it, but this almost always requires major revisions and redisign. An intelligent human is much more autonomous. He can often solve a new kind of problem himself, or find how to proceed by asking someone else or by reading a book.

One might try to explain this by supposing that we have "better thinking processes" than do our programs. But it is premature, we think, to propose a sharp boundary between any of these:

> Having knowledge about how to solve a problem,
>
> Having a procedure that can solve the problem,
>
> Knowing a procedure that can solve the problem!

In any case, we think that much of that a person can do is picked up from his culture in various ways, and the "secrets" of how knowledge is organized lie largely outside the individual. Therefore, we have to find adequate models of how knowledge systems work, how they are acquired by individuals, and how they interact both in the culture and within the individuals.

How can we build programs that need not be rebuilt whenever the problems we want to solve are slightly changed? One wants something less like ordinary computer "programming" and more like "telling" someone how to do something, by informal explanations and examples.

In effect, we want larger effects while specifying less. We do not want to be bothered with "trivial" details. The missing information has to be supplied from the machine's internal knowledge. This in turn requires the machine itself to solve the kinds of easy problems we expect people to handle routinely -- even unconsciously -- in everyday life. The machine must have both the kinds of information and the kinds of reasoning abilities that we associate with the expression "common sense".

There are differences of opinion about such questions, and we digress to discuss the situation. Artificial Intelligence, as a field of inquiry has been passing through a serious crisis of identity. As we see it, the problem stems from the tendency for the pursuit of technical methods to become detached from their original goals so that they follow a developmental pattern of their own. This is not necessarily a bad thing; many productive areas of research were born of such splits. Every discipline has had to deal with such situations and it has happened often in the study of human intelligence. Nevertheless, if one is interested in the particular goal of building a science of intelligence, one has to be concerned with the use of resources both on the local scale of conserving one's own time and energy and on a global scale of watching to see whether the scientific community seems to be directing itself effectively. We suspect that there is now such a problem in connection with the studies of Mechanical Theorem Proving.

5.1 Uniform Procedures Vs. Heuristic Knowledge

As a first approximation to formulating the issues, consider a typical research project working on "automatic theorem proving". Schematically, the project has the form of a large computer program which can accept a body of knowledge or "data base," such as a set of axioms for group theory, or a set of statements about pencils being at desks, desks being in houses, and so on. Given this, the program is asked to prove or disprove various assertions. What normally happens is that if the problem is sufficiently simple, and if the body of knowledge is sufficiently restricted in size, or in content or in formulation, the program does a presentable job. But as the restrictions are relaxed it grinds to an exponential stop of one sort or another.

There are two kinds of strategy for how to improve the program. Although no one actually holds either policy in its extreme form and although we encounter theoretical difficulties when we try to formalize them, it nevertheless is useful to identify their extreme forms.

The POWER strategy seeks a generalized increase in computational power. It may look toward new kinds of computers ("parallel" or "fuzzy" or "associative" or whatever) or it may look toward extensions of deductive generality, or information retrieval, or search algorithms -- things like better "resolution" methods, better methods for exploring trees and nets, hash-coded triplets, etc. In each case the improvement sought is intended to be "uniform" -- independent of the particular data base.

The KNOWLEDGE strategy sees progress as coming from better ways to express, recognize, and use diverse and particular forms of knowledge. This theory sees the problem as epistemological rather than as a matter of computational power or mathematical generality. It supposes, for example, that when a scientist solves a new problem, he engages a highly organized structure of especially appropriate facts, models, analogies, planning mechanisms, self-discipline procedures, etc. To be sure, he also engages "general" problem-solving schemata but it is by no means obvious that very smart people are that way directly because of the superior power of their general methods -- as compared with average people. Indirectly, perhaps, but that is another matter: a very intelligent person might be that way because of specific local features of his knowledge-organizing knowledge rather than because of global qualities of his "thinking" which, except for the effects of his self-applied knowledge, might be little different from a child's.

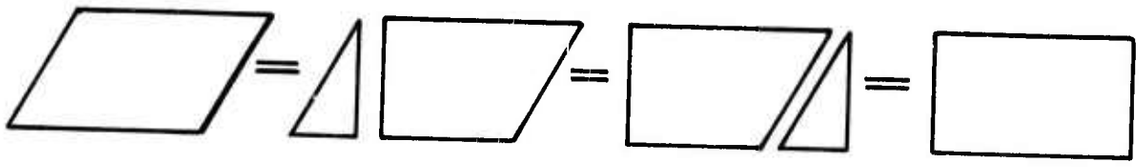
This distinction between procedural power and organization of knowledge is surely a caricature of a more sophisticated kind of "trade-off" that we do not yet know how to discuss. A smart person is not that way, surely, either because he has luckily got a lot of his information well organized or because he has a very efficient deductive scheme. His intelligence is surely more dynamic in that he has (somehow) acquired a body of procedures that guide the organization of more knowledge and the formation of new procedures, to permit bootstrapping. In particular, he learns many ways to keep his "general" methods from taking elaborate but irrelevant deductions and inferences.

5.1.1 Successive Approximations and Plans

The mechanical theorem-proving programs fail unless provided with carefully formulated diets of data; either if given to little knowledge and asked advanced theorems, or given too much knowledge and asked easy questions. In any case, the contrast with a good mathematician's behavior is striking; the programs seem to have no "global" strategies. If a human mathematician is asked to find the volume of some object of unusual shape he will probably try to use some heuristic technique like:

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

Thus, one would transform:



Now, in his final "proof" the heuristic principle that was used will not appear explicitly, even though its use was crucial. The three kinds of information in

1. The knowledge exhibited in the proof;
2. The knowledge used to find the proof;
3. The knowledge required to "understand" or explain the proof so that one can put it to other uses,

are not necessarily the same in extent or in content. The "Theorem Prover" systems have not been oriented toward making it easy to employ the second and the third kinds of knowledge. We have just given an example of how the second type of knowledge can be used.

The third kind of knowledge is exemplified by the following story about an engineer or physicist analyzing a physical system. First, he will make up a fairy-tale:

> "The system has perfectly rigid bodies, that can be treated as purely geometric. There is no friction, and the forces obey Hooke's law."

Then he solves his equations. He finds the system offers infinite resistance to disturbance at a certain frequency. He has used a standard plan; call it ULTRASIMPLE; and it produced an absurdity. But he does not reject this absurdity, completely! Instead, he says: "I know this phenomenon! It tells me that the "real" system has an interesting resonance near this frequency". Next, he calls upon some of his higher-order knowledge about the behavior of plan ULTRASIMPLE. Accordingly, this tells him next to call upon another plan, LINEAR, to help make a new model which includes certain damping and coupling terms.

Next, he studies this system near the interesting frequency that was uncovered by plan ULTRASIMPLE. He knows that his new model is probably very bad at other, far-away, frequencies at which he will get false phenomena because of the unaltered assumptions about rigidity; he has reason to believe these harmless in the frequency band now being studied. Then he solves the new second-order equations. This time he might obtain a pair of finite, close-together resonances of opposite phase. That "explains" the singularity in the simpler model. We abandoned one simple "micro-world" and adopted another, slightly more complicated and better adapted to the better-understood situation. This too may serve only temporarily and then be replaced by a more specialized set of assumptions for studying how nonlinearities affect the fine-structure of the resonances; a new plan, NONLINEAR or INELASTIC or THIRD-ORDER or DISCRETE, or whatever his third-type knowledge suggests.

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

5.2 Micro-worlds and Understanding

In order to study such problems, we would like to have collections of knowledge for several "micro-worlds", ultimately to learn how to knit them together. Especially, we would like to make such a system able to extend its own knowledge base by understanding the kinds of information found in books. One might begin by studying the problems one encounters in trying to understand the stories given to young children in schoolbooks. Any six-year-old understands much more about each of such crucial and various things as

time	space	planning	explaining
causing	doing	preventing	allowing
failing	knowing	intending	wanting
owning	giving	breaking	hurrying

than do any of our current heuristic programs. Eugene Charniak, a graduate student, is now well along in developing some such models, and part of the following discussion is based on his experiences.

Although we might describe this project as concerned with "Understanding Narrative", -- of comprehending a story as a sequence of statements as read from a book -- that image does not quite do justice to the generality of the task. One has the same kinds of problems in:

1. making sense of a sequence of events one has seen or otherwise experienced (what caused what?)

2. watching something being built (why was that done first?)

3. understanding a mathematical proof (what was the real point, what were mere technical details?)

Many mental activities usually considered to be non-sequential have similar qualities, as in seeing a scene: why is there a shadow here? -- What is that? -- Oh, it must be the bracket for that shelf.

In any case, we do not yet know enough about this problem of common sense. One can fill a small book just describing the commonsense knowledge needed to solve an ordinary problem like how to get to the airport, or how to change a tire. Each new problem area fills a new catalogue. Eventually, no doubt, after one accumulates enough knowledge, many new problems can be understood with just a few additional pieces of information. But we have no right to expect this to happen before the system contains the kind of breadth of knowledge a young person attains in his elementary school years!

We do not believe that his knowledge can be dumped into a massive data base without organization, nor do we see how embedding it in a uniformly structured network would do much good. We see competence as emerging from processes in which some kinds of knowledge direct the application of other kinds in which retrieval is not primarily the result of linked associations but rather is computed by heuristic and logical processes that embed specific knowledge about what kinds of information are usually appropriate to the particular goal that is current.

We already know some effective ways to structure logically deep but epistemologically narrow bodies of knowledge, as the result of research on special purpose heuristic programs like MACSYMA, DENDRAL, CHESS, or the Vision System to get experience with broader, if shallower, systems we plan to build up small models or real world situations; each should be a small but complete heuristic problem solving system, organized so that its functions are openly represented in forms that can be understood not only by programmers but also by other programs. Then the simple-minded solutions proposed by these mini-theories may be used as plans for more sophisticated systems, and their programs can be used as starting points for learning programs that intend to improve them.

In the next section we will describe a micro-world whose subject matter has a close relation to the vision world already described. Its objects are geometric solids such as rectangular blocks, wedges, pyramids, and the like. They are moved and assembled into structures by ACTIONS, which are taken on the basis of deductions about such prpperties as shape, spatial relations, support, etc. These interact with a base of knowledge that is partly permanent and partly contingent on external commands and recent events.

5.3 Winograd's BLOCKS World

Note: Sections 5.3 through 5.6 are largely adapted from Terry Winograd's Thesis, but he is not responsible for the oversimplifications and reinterpretations.

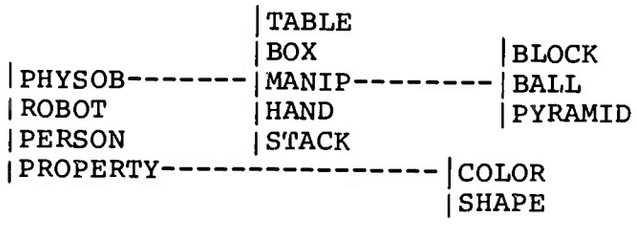
For developing and demonstrating his ideas about understanding natural language, Terry Winograd needed a micro-world in which to carry on a discourse containing statements, questions and commands. In this world we pretend we are talking to a very simple type of robot, like the ones being developed in AI projects at Stanford and MIT. The robot has an arm and an eye. It can look at a scene containing toy objects and can move them with its hand. Winograd did not try to use an actual robot or to simulate it in great physical detail. His "robot" exists only as a display on the CRT scope attached to the computer.

A subject for such a discourse needs a certain amount of structure to support interesting description and manipulation problems. The BLOCKS WORLD has OBJECTS, RELATIONS (and properties) of the objects, ACTIONS that can be performed, and GOALS -- descriptions of states of the world that one might want to achieve.

5.3.1 Objects

In Winograd's model, the robot (named :SHRDLU) has a hand (:HAND) which manipulates objects on a table (:TABLE) that has on it a box (:BOX). The rest of the physical objects are toys -- mainly blocks and pyramids. We give them the names :B1, :B2, "B3, etc. Any symbol beginning with ":" represents a specific object.

Built into this world are some concepts we will use to describe these objects and their properties. We represent them in a tree:



The symbol PHYSOB stands for "physical object" and MANIP for "manipulable object" (i.e. something the robot can pick up). Using the concept IS to mean "has as its basic description," we can write assertions like

(IS :SHRDLU ROBOT) (IS :HAND HAND) (IS :B5 PYRAMDD)

For other, less basic properties we can write attribute-value statements like (MANIP :B5) and (PHYSOB :TABLE). Shape and color are handled with possible shapes are ROUND, POINTED, and RECTANGULAR, and the colors are BLACK, RED, WHITE, GREEN and BLUE. The property names themselves can be treated as objects, so we can make such assertions as (IS BLUE COLOR) and (IS RECTANGULAR SHAPE).

Size and location are more complex, as they depend on the way we choose to represent physical space. We adopted a standard three-dimensional coordinate system and make the simplifying assumption that objects are not allowed to rotate, and therefore always keep their orientation aligned with the coordinate axes. We can represent the position of an object by giving the coordinates of its front lower left-hand corner, and its size by giving three dimensions, as in (AT :B5 (400 600 200)), and (SIZE :B5 (100 100 300)).

5.3.2 Relations

Since we are interested in building structures with the objects around in the scene, one of the most important relations is SUPPORT. The initial data base contains assertions about all of the support relations in the intial scene, like (SUPPORT :B1 :B2). Every time an object is moved, a PLANNER "antecedent theorem: removes the old assertion about what was supporting it, and puts in the correct new one. See 5.3.3. An "antecedent theorem" can be regarded as a sort of demon that watches for some sort of event to happen, and then takes a suitably programmed action. The Blocks World uses a notion of "support" in which an object is supported by whatever is directly below its center of gravity, at the level of its bottom face. Therefore, one object can support several others, but there is only one thing supporting it. Of course this is an extreme simplification since it does not recognize that a simple bridge is supported.

The assertion (CLEARTOP X) will be in the data base if and only if there is no assertion (SUPPORT X Y) for any object Y. Whenever an object is moved, a procedure checks to see if the CLEARTOP status of any object has changed, and if so the necessary erasures and assertions are made.

Information about what is contained in the box is also kept current by an antecedent theorem concerned with the property CONTAIN. The property GRASPING is used to indicate what object (if any) the robot's hand is currently grasping.

Another relation is the PART relation between an object and a stack. We can give a name to a stack, such as :S1, and assert (PART :B2 :S1). Relations using the symbols RIGHT, BEHIND and ABOVE represent the difference in coordinate axes for X, Y and Z respectively. The symbol ON is used to represent the transitive closure of SUPPORT. That is, Z is ON A if A supports B, B supports C, ... supports Z.

The measurements of LENGTH, WIDTH and HEIGHT are contained in the SIZE assertions and (HEIGHT X) evaluates to the height of whatever object the variable X is bound to. SIZE is used for comparisons like "bigger." Currently, it returns the sum of the X, Y and Z coordinates, but it could be easily changed to be more in accord with human psychology. In order to compare measurements, we have the relation MORE. The sentence ":B1 is shorter than "B2" is equivalent to the assertion (MORE HEIGHT :B2 :B1).

OWN relates a person to any object. Knowledge about what the human user owns is gathered from his statements. The semantic programs can use statements about owning to generate further PLANNER theorems which are used to answer questions about what :FRIEND (the human operator) owns and make deductions needed to carry out commands involving owning. The current system contains only token examples of such properties unrelated to the microworld of blocks.

5.3.3 Actions

The only events that can take place in our world are actions taken by the robot in moving its hand and manipulating objects. At the most basic level, the only actions which "really" occur are !MOVETO!, !GRASP! and !UNGRASP!. These are the actual commands sent to the display routines and, theoretically, to a physical robot system.

> To explain how the actions are programmed, in Winograd's system, we need to know a little about the PLANNER language of Carl Hewitt. Briefly, PLANNER has several ways for handling information of the form "A implies B", customarily called "theorems". In one form, the "consequent" form, it is interpreted roughly as "If you want something of the form B, make A a subgoal". In another, the "antecedent" form, it means "if something of the form A occurs, then deduce B and add it to the data base". Still another form of theorem can erase statements, such as support assertions that ought to be excised automatically when one of the participating objects is moved.

The result of calling a consequent theorem to achieve a goal requiring motion, like (PUTON :B3 :B4), is a plan -- a list of instructions using the three elementary functions. !MOVETO! moves the hand and whatever it is grasping to a specified position. !GRASP! sets an indicator that the grasped object is to be moved along with the hand, and !UNGRASP! unsets it. The robot grasps by moving its hand directly over the center of the object on its top surface, and turning on a "magnet." It can do this to any manipulable object, but can only grasp one thing at a time. Using these elementary actions, we can build a hierarchy of actions, including goals that may involve a whole sequence of deductions and actions, like STACKUP which causes the construction of a whole stack of blocks).

Inside the system are another set of "conceptual actions" MOVEHAND, GRASP and UNGRASP, and corresponding consequent theorems to achieve them. There is a significant difference between these and the functions listed above. Calling the function !MOVETO! actually causes the hand to move. On the other hand, when PLANNER evaluates a statement like:

(GOAL(MOVEHAND (600 200 300))(USE tc-MOVEHAND))

nothing is actually moved. Translation: If your goal is to move the hand to (600, 200, 300), use the advice in the tc-MOVEHAND theorem to achieve this goal. The "USE" clause is a feature in PLANNER to allow the insertion of advice on how to achieve goals, etc., in any assertion or theorem. Here, the tc-MOVEHAND theorem creates a plan to do the motion, but if this move would cause us to be unable to achieve a goal at some later point, the PLANNER backup mechanism will automatically erase it from the plan. The robot plans its entire sequence of actions before actually moving anything, trying if necessary all of the recommended means it has to achieve its goal. We do not have space to explain PLANNER's backup system in complete detail; it is described in Hewitt's thesis, and the following sections show roughly how it provides automatic tree searching when necessary, under the control of the "USE" recommendations attached to the theorems in the data base.

These theorems also do some checking to see if we are trying to do something impossible. For example, MOVEHAND makes sure the action would not place one block where there is already an other, and UNGRASP fails unless something will support the object it wants to let go of.

These are the basic objects, relations and actions in the blocks world. But a micro-world also needs concepts about intentions, processes, strategies, etc. We next describe the

strategies programmed into the system for obeying commands about complex goals and for answering questions about its performance and about its intentions.

5.3.4 Carrying Out Commands

Some theorems, like tc-GRASP, are complex, as they can cause a series of actions. The following program gives simplified definitions of various PLANNER theorems. Using these definitions, we will be able to follow the system through a complex action in detail. (For purposes of clarity, the program may be found on the following page).

```
tc-CLEARTOP X
          GO      (COND ((GOAL (SUPPORT X _Y))
                         (GOAL (GET-RID-OF Y) (USE tc-GET-RID-OF))
                         (GO GO))

                        ((ASSERT (CLEARTOP X))))

tc-GET-RID-OF X
                  (OR
                   (GOAL (PUTON X :TABLE)(USE tc-PUTON))
                   (GOAL (PUTON X Y)(USE tc-PUTON)))

tc-GRASP X
                   (GOAL(MANIP X))
                   (COND ((GOAL (GRASPING X)))
                          ((GOAL (GRASPING _Y))
                           (GOAL (GET-RID-OF Y) (USE tc-GET-RID-OF)))
                          (T))
                   (GOAL (CLEARTOP X) (USE tc-CLEARTOP))
                   (SETQ _Y (TOPCENTER X))
                   (GOAL (MOVEHAND Y)
                          (USE tc-MOVEHAND))
                   (ASSERT (GRASPING X))

tc-PUT X
                   (CLEAR Y (SIZE X) X)
                   (SUPPORT Y (SIZE X) X)
                   (GOAL (GRASP X) (USE tc-GRASP))
                   (SETQ _Z (TCENT Y (SIZE X)))
                   (GOAL (MOVEHAND Z)  (USE tc-MOVEHAND))
                   (GOAL (UNGRASP)  (USE tc-UNGRASP)))

tc-PUTON X Y
                   (NOT EQ X Y))
                   (GOAL (FINDSPACE Y $E (SIZE X) X _Z)
                          (USE tc-FINDSPACE tc-MAKESPACE))
                   (GOAL (PUT X Z)  (USE tc-PUT))
```

Let us trace, for example, the meaning of PUTON. The first clause

```
(PUTON X Y)
```

is the "pattern" of the goal. X and Y are variables to be matched. If the goal has this form, then these variables are bound to what they matched and

```
(NOT (EQ X Y))
```

checks for the (impossible) situation of trying to put a block on itself. If this "failure" occurs then the current goal will be abandoned. This means that PLANNER will back up -- reconstruct the situation at the most recent previous variable-binding decision. For example, in this case, the system must have been looking for a place to put the block X, and stupidly decided to put it on X! Now it must make another choice, and presumably this time Y will be found to a different, more sensible location. So this time tc-PUTON will pass the (NOT (EQ X Y)) test and go

on to the next step, which is to create a subgoal:

```
(GOAL (FINDSPACE Y $E (SIZE X) X _Z)
      (USE tc-FINDSPACE tc-MAKESPACE))
```

which says to try to find a space on Y big enough for X, ignoring space currently occupied (possibly) by X. The location resulting from success of this goal is then bound to Z. Again, if the goal fails, we would back up, but the program makes two recommendations for how to find such a place. tc-FINDSPACE says to try to find a space already there; if this fails then tc-MAKESPACE says to try to make such a space.

```
(GOAL (PUT X Z) (USE tc-PUT)))
```

Assuming that this succeeds, then try to use tc-PUT to actually put X in that location Z.

With this explanation we can follow what happens if PLANNER tries the goal:

```
(GOAL (GRASP :B1)(USE tc-GRASP))
```

The theorem tc-GRASP checks to make sure :B1 is a graspable object by looking in the data base for (MANIP :B1). If the hand is already grasping the object, it has nothing more to do. If not, it must first get the hand to the object. This may involve complications -- the hand may already be holding something, or there may be objects sitting on top of the one it wants to grasp. In the first case, it must get rid of whatever is in the hand, using the command GET-RID-OF.

The easiest way to get rid of something is to set it on the table, so tc-GET-RID-OF creates the goal (PUTON X :TABLE), where the variable X is bound to the object the hand is holding. Then tc-PUTON must in turn find a big enough empty place to set down its burden, using the command FINDSPACE, which performs the necessary calculations, using information about the sizes and locations of all the objects. tc-PUTON then creates a goal using PUT, which calculates where the hand must be moved to get the object into the desired place, then calls MOVEHAND to actually plan the move. If we look at the logical structure of our active goals at this point, assuming that we want to grasp :B1, but were already grasping :B2, we see:

```
(GRASP :B1)
      (GET-RID-OF :B2)
        (PUTON :B2 :TABLE)
              (PUT :B2 (453 101 0))
                (MOVEHAND (553 301 100))
```

After moving, tc-PUTON calls UNGRASP, and we have achieved the first part of our original goal -- emptying the hand. Now we must clear off the block we want to grasp. tc-GRASP sets up the goal:

```
(GOAL(CLEARTOP :B2)(USE tc-CLEARTOP))
```

This is a good example of the double use of PLANNER goals to both search the data base and carry out actions. If the assertion (CLEARTOP :B1) is present, it satisfies this goal immediately without calling the theorem. However if :B1 is not already clear, this GOAL statement calls tc-CLEARTOP which takes the necessary actions. Then tc-CLEARTOP will try to GET-RID-OF the objects on top of :B1. This will in turn use PUTON, which uses PUT. But tc-PUT may have more to do this time, since the hand is not already grasping the object is has to move. It therefore sets up a goal to GRASP the object, recursively calling tc-GRASP again.

And so on! To answer questions about the past, the BLOCKS programs remember parts of their subgoal tree by creating objects called events. The system does not remember small, specific steps like MOVEHAND, but only larger goals like PUTON and STACK-UP. The time of events is measured by a clock which starts at 0. It is incremented by 1 every time any motion occurs, creating a new event that combines the original goal statement with an arbitrary name, the starting time, ending time, and "reason" for each event. The reason is the name of the event nearest up in the subgoal tree which is being remembered. (The reason for goals called by the linguistic part of the system is "because you asked me to").

A second kind of memory keeps track of the actual physical motions of objects, noting each time one is moved, and recording its name and the location it went to. This list can be used to establish where any object was at any past time.

When we want to pick up block :B1, we can say: (GOAL(PICKUP :B1)), and it is interpreted as a command. We can also ask "Did you pick up :B1?", since when the robot picked it up an assertion like (PICKUP E2 :B1) was stored in the data base. If PLANNER evaluates PICKUP X :B1)) it will find the assertion, binding the variable X to the event name E2. Since the property list of E2 gives its starting and ending times, and its reason, this is sufficient information to answer most questions.

5.4 Sample Dialog with Winograd's Language Understanding System

To make a useful micro-world one needs not only objects, properties and actions, but also goals, processes, and simplified versions of general concepts like time, space, and intention. The Blocks World was designed by Terry Winograd to serve as a subject domain for the development of a language understanding system. This system is radically more ambitious than earlier attempts to deal with natural language in several respects. It is as much, or more, concerned with the problem of representing the meanings in a discourse than with the grammatical structure; Winograd's thesis asserts that the interactions between meanings and syntactic structures are so intimate that it is a bad idea to try to draw a definite boundary between the procedures that "parse" a sentence and the procedures that are to be used to draw logical inferences about the meanings carried in the discourse.

Winograd's system is described in his thesis at several levels of detail and it is impossible to do justice to it in any short summary. Our strategy will be first to present an example of its performance in the blocks world context, and then to discuss some of the mechanisms that are involved in attaining this performance. In the following dialog the system carries out actions and answers questions about a simple scene containing a table, hand, and box, and several blocks and pyramids, as shown in Fig. 1. The lines in lower case were typed in by a person, and the lines in upper case are the computer's response. Indented lines are comments describing some of the features being exhibited.

The dialog was carried out in real time with the response of the "robot" displayed on a CRT screen. Figures are included to show the state of the scene at various points. It is important to read the dialog with some skepticism. While the apparent linguistic fluency is to an extent genuine, the vocabulary and semantic base is very thin, and the conversation works because the human operator stays closely within the boundaries of the system's understanding.

1. pick up a big red block.
======== OK.

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

2. grasp the pyramid.
======= I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.

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

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

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

======= BY"IT", I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING. (Printed out to keep us informed.)
======= OK.

The hand has to set down the red block to pick up the taller blue one. Figure 1 shows the result of the system's manipulations so far.

4. what does the box contain?
======= THE BLUE PYRAMID AND THE BLUE BLOCK.

5. what is the pyramid supported by?

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

======= THE BOX.

Pick up a big red block

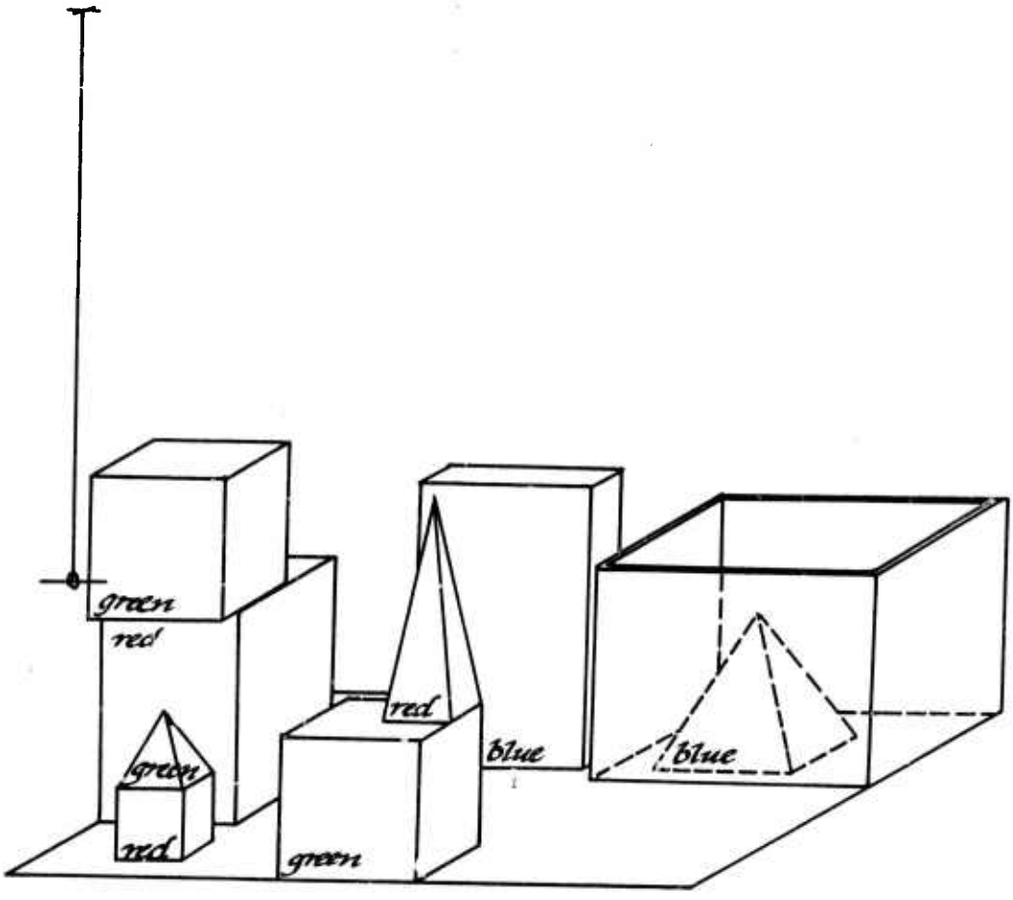


Fig. 1.

6. how many blocks are not in the box?
Negatives such as "not" and "no" are handled whether they appear in clauses or noun groups (such as "no boxes").
======= FOUR OF THEM.

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

8. is it supported? Passives can be used without a "by ..." phrase.
======= YES, BY THE TABLE.

9. can the table pick up blocks?
======= NO. The semantic rules make this a nonsensical question which therefore must be false.

10. can a pyramid be supported by a block?
======= YES. The deductive system finds an actual example, so it knows this is possible.

11. can a pyramid support a pyramid?
======= I DON'T KNOW.
The question makes sense, but there is no example in the scene, so there is no easy way to tell without being able to examine the robot's programs.

12. stack up two pyramids.
======= I CAN'T The robot tries and fails.

13. the blue pyramid is nice.
======= I UNDERSTAND.
This data is added to the data base.

14. i own blocks which are not red, but i don't own anything which supports a pyramid.
======= I UNDERSTAND.
Two new theorems are created for proving things about "owning".

15. do i own the box?
======= NO. The box supports whatever it contains, including a pyramid.

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

17. will you please stack up both of the red blocks and wither a green cube or a pyramid?
======= OK. Logical connectives such as "and" "or" "either" etc. are handled in both the grammar and semantics. A command is executed even if it looks like a question. The command is carried out by the following series of steps: It puts a green cube on the large red block (note that it chooses the green cube with nothing on it), then removes the small pyramid from the little red cube, so that it can use that cube to complete the stack as specified, resulting in the situation of Figure 2.

18. which cube is sitting on the table?
======= THE LARGE GREEN ONE WHICH SUPPORTS THE RED PYRAMID. Ordinarily the system would be satisfied with calling this "a big green cube", but since we asked "which cube" it has to be more specific in naming it. The use of substitute nouns (such as "one") and pronouns in answers makes the discourse more natural.

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

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

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

22. how many things are on top of green cubes?
======= I'M NOT SURE WHAT YOU MEAN BY "ON TOP OF" IN THE PHRASE "ON TOP OF GREEN CUBES". DO YOU MEAN:
1 - DIRECTLY ON THE SURFACE 2 - ANYWHERE ON TOP OF ?
When there are several interpretations and none is clearly better, the system has to ask the user for more information. The paraphrases were included with the dictionary definition of "on", and the request for more information occurs automatically when the heuristics for resolving ambiguities aren't sufficient. A request like this can also occur when the meaning of a pronoun such as "it" or "they" is unclear.

Pick up a big red block.

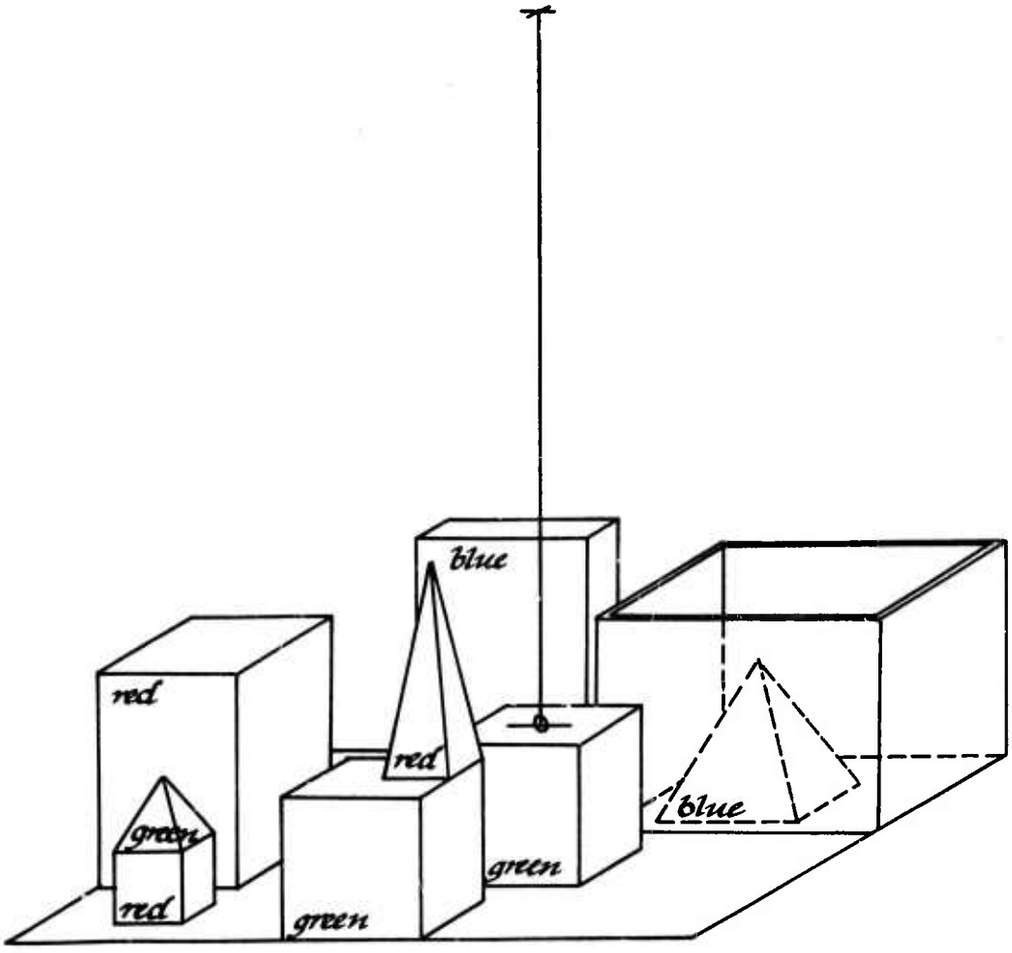


Fig. 2.

2 ====== THREE OF THEM.

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

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

25. why?
The system remembers immediately previous sentences to understand questions like this and the following ones.
======= TO GET RID OF IT.
By keeping track of selected parts of the original subgoal tree, the system has some understanding of its own motives.

26. why did you do that?
======= TO CLEAN OFF THE RED CUBE.
"That" refers to the action mentioned in the answer.

27. why did you clear off that cube?
Here, "that" means the one just referred to.
======= TO PUT IT ON A LARGE GREEN CUBE.

28. why did you do that?
======= TO STACK UP THE RED CUBE, A LARGE RED BLOCK AND A LARGE GREEN CUBE.

29. why did you do that?
======= BECAUSE YOU ASKED ME TO.
It remembers what it did, not how the request was worded.

30. how did you do it?
"It" refers to the action mentioned in the last question, rather than the answer.
======= BY PUTTING A LARGE RED BLOCK ON THE TABLE; THEN LETTING GO OF IT: THEN PUTTING A LARGE GREEN CUBE ON IT: THEN LETTING GO OF THAT CUBE: THEN PUTTING THE RED CUBE ON THAT CUBE: THEN LETTING GO OF THAT CUBE.
This is a description of the top-level subgoals used to achieve the goal.

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

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

======= FOUR OF THEM.

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

======= THE GREEN PYRAMID.

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

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

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

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

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

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

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

======= OK.

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

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

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

36. does a steeple SORRY, I DON'T KNOW THE WORD "STEEPLE".

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

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

The quote marks indicate we are defining a new word.

======= I UNDERSTAND

38. are there any steeples now?
======= NO.

39. build one.

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

40. call the biggest block "superblock".
======= OK. We can also give objects names.

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

42. why did you drop it?
======= BY "IT", I ASSUME YOU MEAN THE SUPERBLOCK.
======= I CAN"T EXPLAIN A NON-EXISTENT EVENT.
There is no record in memory of any block being dropped.

43. is there anything to the right of the red pyramid?
======= YES, FOUR THINGS: THE BOX, THE RED CUBE, THE BLUE PYRAMID AND THE BLOCK NAMED SUPERBLOCK.
Once an object has a name, the name is used to help identify it in responses.

44. thank you.
======= YOU'RE WELCOME!
Idioms can be defined without modifying the grammar.

5.5.1 Syntax and meaning

What concepts are involved in the dialog example? From the performance, it is clear that the system embodies a certain amount of knowledge about blocks, about building, about goals, time, space, words, semantics, syntax, idioms, and common sense about questions, logical inference, tolerance of false assertions, inconsistencies, failures to achieve goals, etc. Each such problem could perhaps be handled, in any particular situation, by appropriate tricks, special case detectors, reduction to standardized schematic situations, etc. but such a system would become more and more limited, unwieldy, and finally incomprehensible and incapable of extension, as situations appear in which special cases interact. In fear of this, perhaps, construction of theories involving meaning has generally been put aside or postponed in favor of attempts to construct syntactic rules that would generate exactly the "grammatical"

sentences of the language. In the work of Chomsky and others it seemed at first that this might work out, but as one attempted more and more realistically comprehensive models, these too turned out to require a great body of special methods, and led to systems that were unwieldy, hard to extend, and finally incomprehensibly complex, just as was feared from the semantic approach. Perhaps, then, the attempt to split syntax completely from semantics actually makes matters worse, and one might do better by facing squarely the problems of meaning!

Such a proposal which once seemed much more difficult than syntactic analysis, now seems easier partly because the latter turned out to be so difficult and partly because advances in heuristic programming made meaning so much less mysterious. It now appears that even a modest semantic complement can greatly simplify understanding syntax.

To emphasize why purely syntactic methods cannot tell us how to parse a sentence and why meaning must be studied, consider the following two sentences:

> The city councilmen refused to give the women a permit for a demonstration because they feared violence.
>
> The city councilmen refused to give the women a permit for a demonstration because they advocated revolution.

If we have to make a choice of who "they" means, for example, to find the gender if we were translating into French, we need the information and reasoning power to realize that city councilmen are usually staunch defenders of law and order, but are hardly likely to be revolutionaries. In traditional syntactic analysis one avoids this problem by announcing both parsings, but if we are interested in understanding how the language is to be used, we have to be able to make the choice. So in addition to a grammar of a language, our program needs all sorts of knowledge about the subject it is discussing, and the ability to use reasoning to combine facts in the right ways. To understand a sentence one has to combine grammar, semantics, and reasoning in a very intimate way, calling on each part to help with the others.

> In earlier computer programs [illegible]erstanding language, such attempts as were made t[illegible] information took the form of lists of rules, pa[illegible] and formulas. In Winograd's system, knowledge is expressed as PROGRAMS in special languages designed to gain the flexibility and power of programs while retaining much of the regularity of traditional simpler rule forms. Since each piece of knowledge can be a procedure, it can call on any other type of knowledge. Thus the "parser" can call semantic programs to see if the phrase it is proposing makes sense, and the semantic programs can call on the deductive programs to see whether that meaning of a phrase makes sense in the current real-world context, as when the choice of a pronoun's assignment depends on the preceding discourse or on detailed knowledge of the subject matter.

While Winograd's system can be described as divided into three parts -- syntax, semantics, and inference -- it is the richness of interplay permitted between these that makes it an advance over previous language-understanding programming attempts. In the following sections we will describe enough of these three "sections" to see how the whole system can handle just the first line in the sample dialog:

pick up a big red block

To fit the type of syntactic analysis he chose to use, Winograd developed a programming language (named PROGRAMMAR) that differs from other parsers in that the grammar is written in the form of a collection of programs. The grammer itself, as we shall explain, is highly suited for semantic analysis since from the start it views the "rules of grammar" as connected with the decisions one makes about conveying meaning rather than about putting words into acceptable orderings.

At the other end of the system we have the knowledge and the reasoning power of a problem-solver system, written in the PLANNER language, to give the system detailed knowledge about its universe -- in this case the BLOCKS WORLD we described in section 5.3. This makes it possible for the system to discuss not only physical happenings but also the robot's own goals and actions.

Interposed between these is the semantic system which contains processes that deduce, from the syntactic constructions, and from the programs that define the meanings of words and other constructions in terms of PLANNER programs, new procedures for the deductive system to use in answering questions, obeying commands, and acquiring new knowledge in the course of the dialog. This system is described in section 5.6. The full system contains some token knowledge also about communication between persons, so that if we say: "There is a block on a green table. What color is it?" the system will assume that "it" refers to the block (rather than the table) since one would not normally ask a question whose answer one knows.

5.5.2 Systemic Grammar

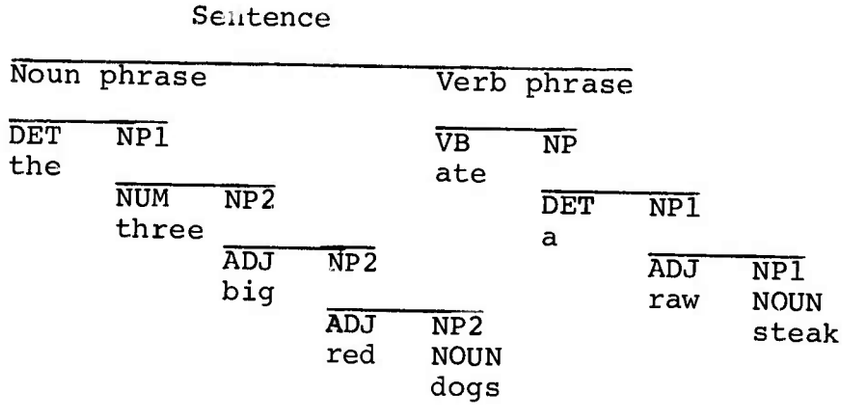
> The following sections might seem unusually detailed for a progress report. But we feel that this system represents a major advance and should be presented in enough detail to see really how it works.

The decision to consider syntax as a proper study devoid of semantics is a basic tenet of most current linguistic theories. Language is viewed as a way of organizing strings of abstract symbols, and tries to explain linguistic competence in terms of symbol-manipulating rules. But although this approach has worked rather well in accounting for which sentences can be formed, it has been unable to shed much light on the basic problem: how does a sentence convey meaning beyond the meanings

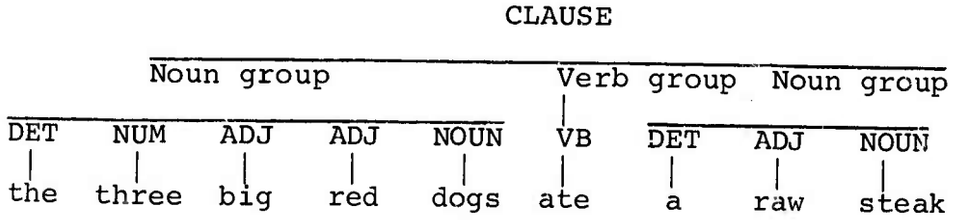
of individual words? Meanings of words depend on other parts of the discourse and intentions depend on one's general orientation and state of knowledge. We can attack the problem in the usual way, by constructing a "mini-theory" as a first approximation, then apply it to see what problems remain.

The structure of a sentence can be viewed as the result of a series of grammatical choices made in generating it. This is not a novel idea in itself; it underlies the most standard notion of generative grammar. But it is not so usual to proceed on to say: the speaker encodes meaning into the sentence by these choices, through choosing to build the sentence with certain "features"; the problem of the hearer is to recognize the presence of those features and interpret their meaning. Of course, we use "feature" to include elements of structural description as well as simple lexicographic terms.

Winograd's system is based on a theory called Systemic Grammar (Halliday, 1967, 1970) which these choices of features are primary. Instead of placing emphasis on a "deep structure" tree, it describes the way different features interact and depend on each other. In other forms of grammar, syntactic structures are usually represented as a binary tree, with many levels of branching and few branches at any node. For example, the sentence "The three big red dogs ate a raw steak." would be parsed with something like this:



Systemic grammar pays more attention to the way language is organized into units, each of which has a special role in conveying meaning. In English we can distinguish three basic ranks of units, the CLAUSE, the GROUP, and the WORD. In systemic grammar, the same sentence might be viewed as having this structure.



In this analysis, the WORD is the basic building block. There are word classes like "adjective", "noun", "verb". The word "dogs" is the same basic vocabulary item as "dog", but has the feature "plural" instead of "singular". "Took", "take", "taken", "taking", etc., are all the same basic word, but with differing features such as "past participle", "infinitive", "-ing", etc.

The next unit above the WORD is the GROUP. Noun groups (NG) describe objects, verb groups (VG) carry complex messages about the time and modal (logical) status of an event or relationship, preposition groups (PREPG) describe certain simple relationships, while adjective groups (ADJG) convey other kinds of relationships and descriptions of objects.

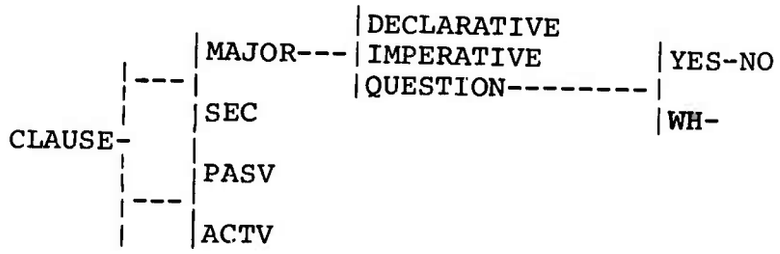
Each GROUP can have "slots" for the words of which it is composed. As we shall see, a NG has slots for "determiner" (DET), "numbers" (NUM), "adjectives" (ADJ), "classifiers" (CLASF), and a NOUN. Each group can also exhibit features, just as a word can. A NG can be "singular" (NS) or "plural" (NPL), "definite" (DEF) as in "the three dogs" or "indefinite" (INDEF) as in "a steak", and so forth. A VG can be "negative" (NEG) or not, can be MODAL (as in "could have seen"), and can have a complex tense.

The CLAUSE is the most complex and diverse unit of the language, and is used to express relationships and events, involving time, place, manner and many other aspects of meaning. It can be a QUESTION, a DECLARATIVE, or an IMPERATIVE, it can be "passive" or "active", it can be a YES-NO question or a WH- question (like "Why...?" or "Which...?"). Our second parsing tree showed how a clause may be composed of groups, which are in turn made up of words. Also, groups often contain other groups; for example, "the call of the wild" is an NG, which contains the PREPG "of the wild" which in turn contains the NG "the wild". Clauses can be parts of other clauses, as in "Join the Navy to see the world.", and can be used as parts of groups in many different ways, as in the NG "the man who came to dinner" or the PREPG "by leaving the country".

If the units can appear anywhere in the tree, what is the advantage of grouping constituents into "units" instead of having a detailed structure like the one shown in our first parsing tree? The answer is that each unit has associated with it a set of meaning-carrying features, related by definite logical structures. The choice between YES-NO and WH- is meaningless unless the clause is a QUESTION, but if it is a QUESTION, the choice must be made.

Similarly, the choice between QUESTION, IMPERATIVE, and DECLARATIVE is mandatory for a MAJOR clause (one which could stand alone as a sentence) but is not possible for a "secondary" (SEC) clause, such as "the country which possesses the bomb." The choice between PASV -- "the ball was attended by John" -- and ACTV -- "John attended the ball" -- is on a totally different dimension, since it can be made regardless of which of these other features are present.

A set of mutually exclusive features like QUESTION, DECLARATIVE, and IMPERATIVE is called a system, and will be diagrammed by connecting them with a vertical bar. Each system has an entry condition which can be an arbitrary boolean condition on the presence of other features. For example, in the diagram below, one of the systems has the feature MAJOR as its entry condition, since only MAJOR clauses make the choice between DECLARATIVE, IMPERATIVE, and QUESTION. We can diagram some of our CLAUSE features as:



The choice between SEC and MAJOR and the choice between PASV and ACTV both depend directly on the presence of CLAUSE. This type of relationship will be indicated by a bracket in place of a vertical bar.

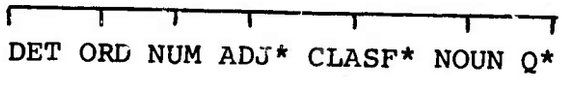
In addition, a syntactic "unit" can have different functions as a part of a larger unit. A transitive clause must have units to fill the functions of SUBJECT and OBJECT, and a WH- question has to have a constituent to play the role of "question element" like "which dog" in "Which dog stole the show?".

In most current theories, there is no explicit mention of these features and functions in the syntactic rules, but the rules are designed in such a way that every sentence will in fact be one of the three types listed above, and every WH- question will in fact have a question element. The difficulty is that there is no attempt in such a grammar to distinguish meaning-conveying features such as these from the many other features we could note about a sentence, and which are also implied by the rules.

5.5.3 The Noun Group

We illustrate these ideas by presenting the structure of the NOUN GROUP in some detail, closely following the presentation in Winograd's thesis.

Here is the structure of the typical NG, using a "*" to indicate that the same element can occur more than once. Most of these "slots" are optional, and may or may not be filled in any particular NG.



The most important ingredient is the NOUN, which gives the basic information about the object or objects being referred to by the NG. Immediately preceding the NOUN, there are an arbitrary number of "classifiers", like "plant life" or "water meter cover and adjustment screw". The same class of words can serve as CLASF and NOUN, in English, and our dictionary gives the meaning of words according to their word class, because nouns often have a special meaning when used as a CLASF.

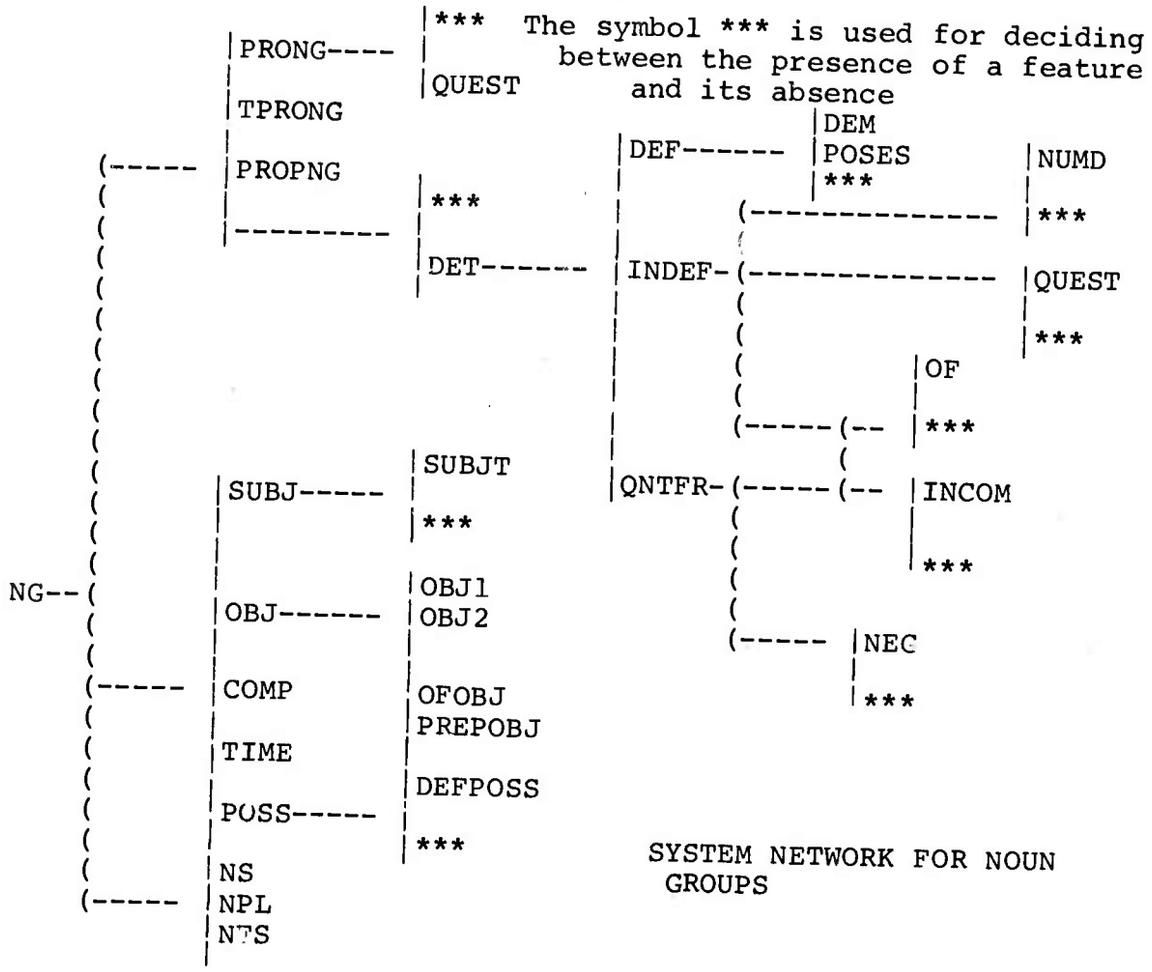
Preceding the classifiers we have adjectives (ADJ) such as "big beautiful soft red". Adjectives can be used as the complement of a BE CLAUSE, but classifiers cannot. We can say "red hair", or "horse hair", or "That hair is red.", but we cannot say "That hair is horse.", since "horse" is a CLASF, not an ADJ. Adjectives can also take on the comparative, superlative forms ("red, reder, and reddest"), while classifiers cannot ("horse, horser, and horsest"?). Immediately following the NOUN we can have various qualifiers (Q), which can be a PREPG like "the man in the moon" or an ADJG like "a night darker than doom" or a CLAUSE RSQ like "the woman who conducts the orchestra".

The first few elements in the NG work together to give its logical description -- whether it refers to a single object, a class of objects, a group of objects, etc. The determiner (DET) is the normal start for a NG, and can be a word such as "a", or "that", or a possessive. It may be followed by an "ordinal" (ORD), as "first, second, third", etc., or a few others such as "last" and "next". These are the only words that can appear between a DET like "the" and a number, as in "the next three days". Finally there is a number (NUM), like "one", "two", etc. or a more complex construction such as "at least three", or "more than a thousand". It is possible for a NG to have all slots filled, as in:

DET ORD NUM ADJ ADJ CLASF CLASF NOUN Q(PREPG)
the first three old red city fire hydrants without

Q(CLAUSE)
covers you can find

With these basic components in mind, let us look at the system network for NG.



SYSTEM NETWORK FOR NOUN GROUPS

At the top of the diagram are some special cases which do not have the structure described above. An NG made up of a pronoun is a PRONG. It can be either a question, like "who" or "what", or a non-question (the unmarked case) like "I", "them", "it", etc. The feature TPRONG marks a NG whose head is a special TPRON, like "something", "everything", "anything", which can enter into a peculiar construction in which an adjective can follow the head, as in "anything green which is bigger than the moon". It has its own special syntax. The feature PROPNG marks an NG made up of proper nouns, such as "Oklahoma", or "The Union Of Soviet Socialist Republics".

The rest of the noun groups are the normal type, discussed above. The DET can be definite (like "the" or "that", indefinite like "a" or "an", or a quantifier (QNTFR) like "some", "every", or "no". The definite determiners can be either demonstrative ("this", "that", etc.) or the word "the" (the unmarked case), or a possessive NG. The NG "the farmer's son" has the NG "the farmer" as its determiner, and has the feature POSES to indicate this.

An INDEF NG can have a number as a determiner: "five gold rings", or "at least a dozen eggs", in which case it has the feature NUMDET, or it can use an INDEF determiner, such as "a". In

either case it has the choice of being a question. The question form of a NUMDET is "how many", while for other cases it is "which" or "what".

Finally, an NG can be determined by a quantifier (QNTFR). Although quantifiers could be subclassified along various lines, we do so in the semantics rather than the syntax. The only classifications used syntactically are between singular and plural, and between negative and non-negative.

If a NG is either NUMD or QNTFR, it can be of a special type marked OF, as in "all of your dreams", but can also choose to be incomplete, leaving out the NOUN, as in "Give me three" or "I want none". There is a correspondence between the cases which can take the feature OF, and those which can be INCOM. We cannot say either "the of them" or "Give me the". Possessives are an exception, we can say "Give me Juan's" but not "Juan's of them", and are handled separately.

The middle part of the NG Network describes the different possible functions an NG can serve. In the CLAUSE, one can use an NG as a SUBJ, COMP, or objects OBJ of various types. In addition, it can serve as the object of a PREPG (PREPOBJ), in: "the rape of the lock". If it is the object of "of" in an OF NG, it is called an OFOBJ: "none of your tricks". An NG can also be used to indicate TIME, as in: "Yesterday the world ended" or "The day she left, all work stopped".

Finally, an NG can be the possessive determiner for another NG. In: "the cook's kettle" the NG "the cook" has the feature POSS, indicating that it is the determiner for the NG "the cook's kettle", which has the feature POSES.

When a PRONG is used as a POSS, it must use a special possessive pronoun, like "my", "your", etc. We can use a POSS in an incomplete NG, like "Show me yours" or "John's is covered with mud". There is a special class of pronouns used in these noun groups (labelled DEFPOSS), such as "yours", "mine", etc.

Continuing to the last part of the NG Network, we see features of person and number. These are used to match the noun to the verb (if the NG is the subject) and the determiner, to avoid combinations like "these kangaroo" or "the women wins". In the case of a PRONG, there are special pronouns for first, second, and third person, singular and plural. The feature NFS occurs only with the first-person singular pronouns ("I", "me", "my", "mine"), and no distinction is made between other persons, since they have no effect on the parsing. A singular pronoun or other singular NG is marked with the feature NS. The pronoun "you" is always treated as if it were plural and no distinction is made between "we", "you", "they", or any plural (NPL) NG as far as the grammar is concerned. Of course there is a semantic difference.

5.5.4 The Parser in Action

With this sketch of some of the ingredients, we can now follow the parser through an example to get a feeling for the way the grammar works, and the way it interacts with the different features described above. Consider the first sentence of our sample dialog.

"Pick up a big red block."

The system begins trying to parse a sentence, which means looking for a MAJOR CLAUSE. It activates the grammar by calling the CLAUSE program with an initial feature list of (CLAUSE MAJOR).

The CLAUSE program looks at the first word, to decide what unit the CLAUSE begins with. If it sees an adverb, it assumes the sentence begins with a single-word modifier; if it sees a preposition, it looks for an initial PREPG. If it sees a preposition, it looks for an initial PREPG. If it sees a BINDER, it calls the CLAUSE program to look for a BOUND CLAUSE. In English (and possibly all languages) the first word of a construction often gives a very good clue as to what that construction will be. In this case, "pick" is a verb, and indicates that we may have an IMPERATIVE CLAUSE. The program starts the VG program with the initial VG feature list (VG IMPER), looking for a VG of this type. This must either begin with some form of the verb "do", or with the main verb itself. Since the next word is not "do", it checks the next word in the input (in this case still the first word) to see whether it is the infinitive form of a verb. If so, it is to be attached to the parsing tree, and given the additional feature MVB (main verb). The current structure can be diagrammed as

```
(CLAUSE MAJOR)
  (VG IMPER)
    (VB MVB INF TRANS VPRT----------pick
```

TRANS and VPRT came from the definition of the word "pick" when we called the function PARSE for a word.

When the VG program succeeds, CLAUSE takes over again. Since it has found the right kind of VG for an imperative CLAUSE, it puts the feature IMPER on the CLAUSE feature list. It then checks to see whether the MVB has the feature VPRT, indicating it is a special kind of verb which takes a particle. It discovers that "pick" is such a verb, and next checks to see if the next word "up" is a PRT, which it is. It then checks in the dictionary and finds out that the combination "pick up" is defined, so it calls (PARSE PRT) to add "up" to the parsing tree. We might have let the VG program do the work of looking for a PRT, but it would have run into difficulties with sentences like "Pick the red block up." in which the PRT is displaced. By letting the CLAUSE program do the looking, the problem is simplified.

As soon as it has parsed the PRT, the CLAUSE program marks the feature PRT on its own feature list. It then looks at the dictionary entry for "pick up" to see what transitivity features are there. It is transitive, which indicates that we should look for one object -- OBJ1. The dictionary entry shows that the object must be either an NG or a WHRS clause (which would begin with a relative pronoun, like "Pick up what I told you to." Since the next word is "a", this is not the case, so the CLAUSE program looks for an object by calling (PARSE NG OBJ OBJ1), asking the NG program to find an NG which can serve as an OBJ1. The structure is now

```
(CLAUSE MAJOR IMPER PRT)
  (VG IMPER)
   (VB MVB INF TRANS PRT)-------------pick
  (PRT)------------------------------  up
  (NG OBJ OBJ1)
```

The NG program notices that the upcoming word is a determiner, "a". It calls (PARSE DET) to add it to the parsing tree, then transfers the relevant features from the DET to the entire NG. It also adds the feature DET to the NG to indicate that it has a determiner. The feature list for the NG is now:

(NG OBJ OBJ1 DET INDEF NS)

since "a" is a singular indefinite determiner. The NG program then notices the feature INDEF, and decides not to look for a number or an ordinal -- we can't say "a next three blocks" -- or for the OF construction -- "a of them" is impossible. It goes on immediately to look for an adjective by calling (PARSE ADJ). When this succeeds with the next word "big", a simple program loop returns to the (PARSE ADJ) statement, which succeeds again with "red". On the next trip it fails, and sends the program on to look for a classifier, since "block" isn't an ADJ. But "block" isn't a CLASF either in our dictionary, so the NG program goes on to look for a NOUN, by calling (PARSE NOUN). This succeeds with the NOUN "block", which is singular, and the program checks to see if it agrees with the number features already present from the determiner (to eliminate illegal combinations like "these boy"). In this case, both are singular (NS), so the program is satisfied. Ordinarily it would go on to look for qualifiers, but in this case there is nothing left in the sentence. Since we have found all of the basic constituents we need for an NG, the NG program should return success. If we had run out after the determiner, it would have checked for an incomplete NG, while if we had run out after an ADJ it would have entered a backup program which would check to see whether it had misinterpreted a NOUN as an ADJ.

In this case, the NG program returns, and the CLAUSE program notices that the sentence has ended. Since a TRANS verb needs only one object, and that object has been found, the CLAUSE program marks the feature TRANS, and returns, ending the parsing. In actual use, a semantic program would be called now to understand and execute the command -- in fact, semantic programs

would have been called at various points throughout the process. The final result is:

```
(CLAUSE MAJOR IMPER PRT TRANS)
  (VG IMPER)
   (VB MVB INF TRANS VPRT)-----------pick
  (PRT---------------------------------- up
  (NG OBJ OBJ1 DET INDEF NS)
   (DET INDEF NS)--------------------a
   (ADJ)-----------------------------big
   (ADJ)-----------------------------red
   (NOUN NS)-------------------------block
```

5.6 Semantic Structures

In 5.5 we described some of the operation of the systemic grammar parsing program. For the semantic system we again will use the Noun Group as an example, to present the general idea. As one hears or reads linguistic sequences, one extracts meanings and uses them to modify one's model of the world or in some other way to organize one's behavior. In Winograd's system, the meanings are usually represented by procedures written in the Planner language. There are a number of ways in which these procedures are used to build up meanings by cooperation between the systemic-grammar analyzer and other processes called "semantic specialists".

One of the most obvious semantic functions of expressions is to describe objects, and the "noun group" is most commonly used for this. It contains a noun which indicates the kind of object, adjectives and classifiers which describe further properties of the object; and a complex system of quantifiers and determiners describing its logical status -- whether it is a particular object, a class of objects, a particular set of objects, or even an unspecified set containing a specified number of objects ("three bananas"), etc. The syntactic structure already discussed provides a systematic framework for such descriptions. One might object that this is too rigid and that there are other ways to describe objects. Indeed, but this one handles a wide range of ordinary cases and Winograd's PROGRAMMAR system supplies an unprecedented flexibility for introducing other methods and even complex heuristic programs for dealing with other situations.

The semantic system is built around a dozen or so programs, "semantic specialists" which are experts at interpreting particular syntactic structures. These are called by PROGRAMMAR when the parsing system believes that a certain structure, say, a noun group, has been parsed. They look at both the syntactic structures and the meanings of the words (which are also represented by programs), and build up PLANNER theorems which can be used either by the deductive mechanisms (for performing actions in, or for answering questions about, the Blocks World) or by the syntactic system itself to decide whether the proposed noun group is meaningful.

A Noun Group like "a red cube" can be described as:

```
(GOAL (IS X BLOCK))
(EQDIM X)
(GOAL (COLOR X RED))
```

The variable "X" represents the object, and this description says that the object X should be a block, it should have equal dimensions, and it should be red. A phrase such as "a red cube which supports three pyramids but is not contained in a box" would be built up from the descriptions for the various objects, and would end up as

```
(GOAL (IS X BLOCK))
(EQDIM X)
(GOAL (COLOR X RED))
(FIND 3 X 2 (GOAL (IS X2 PYRAMID))
            (GOAL (SUPPORT X X2)))
(NOT (PROG X3
     (GOAL (IS X3 BOX))
     (GOAL (CONTAIN X3 X))))
```

This "meaning" is a procedure. A larger deductive system could use it

to find such an object;

to say whether one exists;

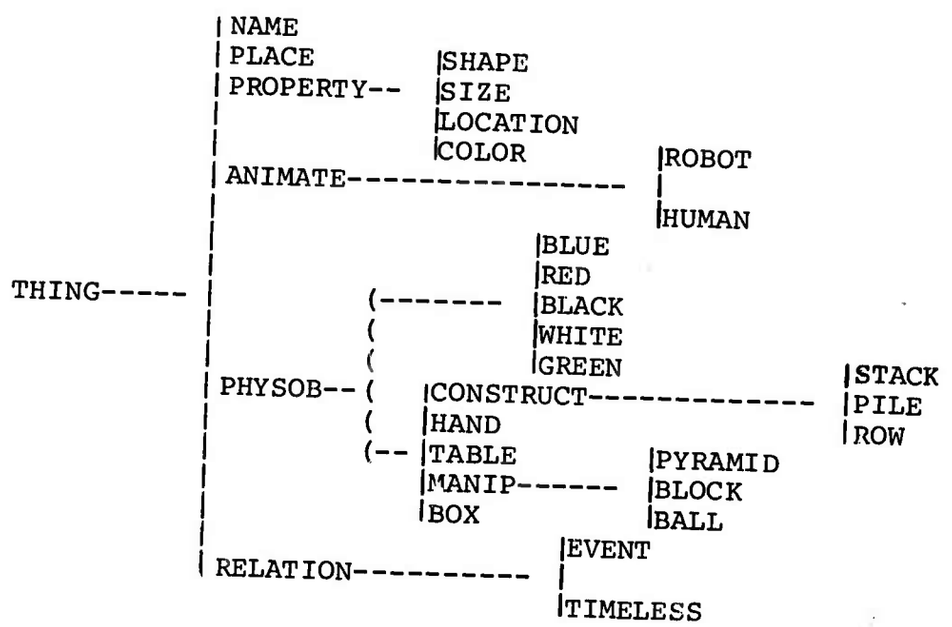
to list relations in which it does, or could, participate;

to answer more abstract questions about whether such an object could exist or (as in the BLOCKS program) to plan a sequence of actions that will cause it to exist.

Furthermore, the "theorem" that embodies the meaning could be used within the parsing process itself, for if the deductive system finds that there could be no such object then the alleged noun group would be suspect and one could search for an alternative parsing. One could imagine a much more sophisticated system that would suspend this strategy if the discourse concerns a subject, like language itself, in which normally unacceptable expressions are sometimes permitted.

How do the semantic specialists build this structure? Consider the simple expression "a red cube". First the noun group is parsed, then the PLANNER description is built up backwards by the specialists, starting with the noun, and continuing in right-to-left order through the classifiers and adjectives.

Part of the definition for a noun uses semantic markers to filter out meaningless interpretations of a phrase. The BLOCKS world uses this tree of semantic markers:



Again, vertical bars represent exclusive choices, while horizontal lines represent logical dependency. "PHYSOB" means "physical object", and "MANIP" means "manipulable object". The first specialist, SMNG1, finds that the definition of the noun "cube" is:

```
(NMEANS (CUBE) ((IS X BLOCK)(EQDIM X)))
```

which says that a cube is a block with equal dimensions. NMEANS is the name of a function for dealing with nouns, which accepts a list of different meanings for a word. In this case, there is only one meaning. When NMEANS is executed, it puts information onto the semantic structure which is being built for the object. It takes care of finding out what markers are implied by the tree, here (THING PHYSOB MANIP BLOCK), deciding which predicates need to be in a GOAL statement (like IS), and which are LISP predicates (like EQDIM). It also can decide on recommendation lists to put onto the PLANNER goals, to guide deductions.

Next, SMNG1 calls the definition for the adjective "red".

```
(NMEANS((PHYSOB)((COLOR X RED))))
```

This definition indicates that the property applies only to physical objects.

There is no absolute definition for "big" or "little"; a "big flea" is still not much competition for a "little elephant". The meaning of the adjective is relative to the noun it modifies, and it may also be relative to the adjectives following it as well, as in a "big toy elephant." As the system analyzes the NG from right to left, the meaning of each adjective is added to the description already built up for the head and modifiers to the right. Since each definition is a

program, it can just as well examine the description (both the semantic markers and the PLANNER description), and produce an appropriate meaning relative to the object being described. This may be an absolute measurement (e.g., a "big elephant" is more than 12 feet tall) or a relative PLANNER description of the form "the number of objects fitting the description and smaller than the one being described is more than the number of suitable objects bigger than it is."

In adding the meaning of "red" to the semantic structure, the specialist must make a choice in ordering the PLANNER expressions. In the robot's tiny world, this isn't of much importance, but if the data base took phrases like "a man in this room", we certainly would be better off looking around the room first to see what was a man, than looking through all the men in the world to see if one was in the room. To make this choice we allow each predicate (like IS or COLOR) to have associated with it a program which knows how to evaluate its "priority" in any given environment. The program might be as simple as a single number, or it might be a complex heuristic program which takes into account the current state of the world and the discourse.

Here is the structure which would be built up by the program.

```
        (GOAL (IS X BLOCK))
        (GOAL (COLOR X RED))
        (EQDIM X)----------------PLANNER description
    (BLOCK MANIP PHYSOB THING)------markers
    (MANIP PHYSOB THING)------------systems
    (NS INDEF)----------------------determiner
```

Let us now take a slightly more complicated NG, "a red cube which supports a pyramid". We can only summarize what happens here. First, the NG parsing program finds the determiner ("a"), adjective ("red"), and noun ("cube"). Then, after further analysis a CLAUSE specialist is called to deal with "which supports a pyramid" and it constructs a corresponding plannar theorem, for the meaning of "a pyramid". Next the definition of the verb "support" is called, and used to build up an assertion that the subject and object are related by SUPPORT.

The clause is now finished, and the specialist on relative clauses (SMRSQ) is called to take the PLANNER descriptions of the objects involved in the relation, along with the relation itself, and put the information onto the PLANNER description of the object to which the clause is being related. The result, for the description of "a red cube which supports a pyramid" is

```
        (GOAL (IS X BLOCK))
        (GOAL (COLOR X RED))
        (EQDIM X)
        (GOAL (IS X2 PYRAMID))
        (GOAL (SUPPORT X X2))------PLANNER description
    (BLOCK MANIP PHYSOB THING)---------markers
    (MANIP PHYSOB THING)---------------systems
    (NS INDEF)-------------------------determiner
```

Relationships have the full capability to use semantic markers just as objects do, and at an early stage of construction, a relation structure contians a PLANNER description, markers, and systems in forms identical to those for object structures (this is to share some of the programs, such as those which check for conflicts between markers). We can classify different types of events and relationships (for example, those which are changeable, those which involve physical motion, etc.) and use the markers to help filter out interpretations of clause modifiers. For example, the modifying PREPG "without the shopping list" in

"He left the house without the shopping list."

has a different interpretation from "without a hammer" in

"He built the house without a hammer.".

If we had a classification of activities which included those involving motion and those using tools, we could choose the correct interpretation.

APPENDIX A

PROJECT MAC TECHNICAL REPORTS*

† TR-1 Bobrow, Daniel G.
Natural Language Input for A Computer Problem Solving System, Ph.D. Thesis, Math. Dept.
September 1964 AD 604-730

† TR-2 Raphael, Bertran
SIR: A Computer Program for Semantic Information Retrieval, Ph.D. Thesis, Math. Dept.
June 1964 AD 608-499

TR-3 Corbató, Fernando J.
System Requirements for Multiple-Access, Time-Shared Computers
May 1964 AD 608-501

† TR-4 Ross, Douglas T., and Clarence G. Feldman
Verbal and Graphical Language for the AED System: A Progress Report
May 6, 1964 AD 604-678

† TR-6 Biggs, John M. and Robert D. Logcher
STRESS: A Problem-Oriented Language for Structural Engineering
May 6, 1964 AD 604-679

† TR-7 Weizenbaum, Joseph
OPL-1: An Open Ended Programming System Within CTSS
April 30, 1964 AD 604-680

† TR-8 Greenberger, Martin
The OPS-1 Manual
May 1964 AD 604-681

† TR-11 Dennis, Jack B.
Program Structure in a Multi-Access Computer
May 1964 AD 608-500

TR-12 Fano, Robert M.
The MAC System: A Progress Report
October 9, 1964 AD 609-296

† TR-13 Greenberger, Martin
A New Methodology for Computer Simulation
October 19, 1964 AD 609-288

TR-14 Roos, Daniel
Use of CTSS in a Teaching Environment
November 1964 AD 661-807

APPENDIX A

TR-16 Saltzer, Jerome H.
CTSS Technical Notes
March 1965
AD 612-702

TR-17 Samuel, Arthur L.
Time-Sharing on a Multiconsole Computer
March 1965
AD 462-158

† TR-18 Scherr, Allan Lee
An Analysis of Time-Shared Computer Systems, Ph.D. Thesis, EE Dept.
June 1965
AD 470-715

TR-19 Russo, Francis John
A Heuristic Approach to Alternate Routine in a Job Shop, S.B. & S.M. Thesis, Aloan School
June 1965
AD 474-018

TR-20 Wantman, Mayer Elihu
CALCULAID: An On-Line System for Algebraic Computation and Analysis, S.M. Thesis, Sloan School
September 15, 1965
AD 474-019

TR-21 Denning, Peter James
Queueing Models for File Memory Operation, S.M. Thesis, EE Dept.
October 1965
AD 624-943

† TR-22 Greenberger, Martin
The Priority Problem
November 1965
AD 625-728

TR-23 Dennis, Jack B. and Earl C. Van Horn
Programming Semantics for Multiprogrammed Computations
December 1965
AD 627-537

† TR-24 Kaplow, Roy, Stephen Strong and John Brackett
MAP: A System for On-Line Mathematical Analysis
January 1966
AD 476-443

TR-25 Stratton, William David
Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays, S.M. Thesis, EE Dept.
March 7, 1966
AD 631-386

TR-26 Cheek, Thomas Burrell
Design of a Low-Cost Character Generator for Remote Computer Displays, S.M. Thesis, EE Dept.
March 8, 1966
AD 631-269

TR-27 Edwards, Daniel James
OCAS - On-Line Cryptanalytic Aid System
S.M. Thesis, EE Dept.
May 1966 AD 633-678

TR-28 Smith, Arthur Anshel
Input/Output in Time-Shared, Segmented, Multiprocessor Systems, S.M. Thesis, EE Dept.
June 1966 AD 637-215

TR-29 Ivie, Evan Leon
Search Procedures Based on Measures of Relatedness Between Documents, Ph.D. Thesis, EE Dept.
June 1966 AD 636-275

TR-30 Saltzer, Jerome Howard
Traffic Control in a Multiplexed Computer System, Sc.D. Thesis, EE Dept.
July 1966 AD 635-966

TR-31 Smith Donald L.
Models and Data Structures for Digital Logic Simulation, S.M. Thesis, EE Dept.
August 1966 AD 637-192

TR-32 Teitelman, Warren
PILOT: A Step Toward Man-Computer Symbiosis, Ph.D. Thesis, Math. Dept.
September 1966 AD 638-446

TR-33 Norton, Lewis M.
ADEPT - A Heuristic Program for Proving Theorems of Group Theory, Ph.D. Thesis, Math. Dept.
October 1966 AD 645-660

TR-34 Van Horn, Earl C.
Computer Design for Asynchronously Reproducible Multiprocessing, Ph.D. Thesis, EE Dept.
November 1966 AD 650-407

TR-35 Fenichel, Robert R.
An On-Line System for Algebraic Manipulation Ph.D. Thesis, Appl. Math. (Harvard)
December 1966 AD 657-282

† TR-36 Martin, Willaim A.
Symbolic Mathematical Laboratory, Ph.D. Thesis, EE Dept.
January 1967 AD 657-283

TR-37 Guzman-Arenas, Adolfo
Some Aspects of Pattern Recognition by Computer, S.M. Thesis, EE Dept.
February 1967 AD 656-041

APPENDIX A

TR-38 Rosenberg, Ronald C., Daniel W. Kennedy and Roger A. Humphrey
A Low-Cost Output Terminal for Time-Shared Computers
March 1967 AD 662-027

TR-39 Forte, Allen
Syntax-Based Analytic Reading of Musical Scores
April 1967 AD 661-806

TR-40 Miller, James R.
On-Line Analysis for Social Scientists
May 1967 AD 668-009

TR-41 Coons, Steven A.
Surfaces for Computer-Aided Design of Space Forms
June 1967 AD 663-504

TR-42 Liu, Chung L., Gabriel D. Chang and Richard E. Marks
Design and Implementation of a Table-Driven Compiler System
July 1967 AD 668-960

TR-43 Wilde, Daniel U.
Program Analysis By Digital Computer, Ph.D. Thesis, EE Dept.
August 1967 AD 662-224

TR-44 Gorry, G. Anthony
A System for Computer-Aided Diagnosis, Ph.D. Thesis, Sloan School
September 1967 AD 662-665

TR-45 Leal-Cantu, Nestor
On the Simulation of Dynamic Systems with Lumped Parameters and Time Displays, S.M. Thesis, ME Dept.
October 1967 AD 663-502

TR-46 Alsop, Joseph W.
A Canonic Translator, S.B. Thesis, EE Dept.
November 1967 AD 663-503

† TR-47 Moses, Joel
Symbolic Integration, Ph.D. Thesis, Math Dept.
December 1967 AD 662-666

TR-48 Jones, Malcolm M.
Incremental Simulation on a Time-Shared Computer, Ph.D. Thesis, Sloan School
January 1968 AD 662-225

APPENDIX A

TR-49 Luconi, Fred L.
Asynchronous Computational Structures, Ph.D. Thesis, EE Dept.
February 1968 AD 677-602

† TR-50 Denning, Peter J.
Resource Allocation in Multiprocess Computer Systems, Ph.D. Thesis, EE Dept.
May 1968 AD 675-554

† TR-51 Charniak, Eugene
CARPS, A Program which Solves Calculus Word Problems, S.M. Thesis, EE Dept.
July 1968 AD 673-670

TR-52 Deitel, Harvey M.
Absentee Computations in a Multiple-Access Computer System, S.M. Thesis, EE Dept.
August 1968 AD 684-738

TR-53 Slutz, Donald R.
The Flow Graph Schemata Model of Parallel Computation, Ph.D. Thesis, EE Dept.
September 1968 AD 683-393

TR-54 Grochow, Jerrold M.
The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System, S.M. Thesis, EE Dept.
October 1968 AD 689-468

TR-55 Rappaport, Robert L.
Implementing Multi-Process Primitives in a Multiplexed Computer System, S.M. Thesis, EE Dept.
November 1968 AD 689-469

† TR-56 Thornhill, D. E., R. H. Stotz, D. T. Ross and J. E. Ward (ESL-R-356)
An Integrated Hardware-Software System for Computer Graphics in Time-Sharing
December 1968 AD 685-202

TR-57 Morris, James H.
Lambda-Calculus Models of Programming Languages, Ph.D. Thesis, Sloan School
December 1968 AD 683-394

TR-58 Greenbaum, Howard J.
A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System, S.M. Thesis, EE Dept.
January 1969 AD 686-988

TR-59 Guzman, Adolfo
Computer Recognition of Three-Dimensional Objects in a Visual Scene, Ph.D. Thesis, EE Dept.
December 1968 AD 692-200

APPENDIX A

† TR-60 Ledgard, Henry F.
A Formal System for Defining the Syntax and Semantics of Computer Languages, Ph.D. Thesis, EE Dept.
April 1969 AD 689-305

TR-61 Baecker, Ronald M.
Interactive Computer-Mediated Animation, Ph.D. Thesis, EE Dept.
June 1969 AD 690-887

† TR-62 Tillman, Coyt C. (ESL-R-395)
EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation
June 1969 AD 692-462

TR-63 Brackett, John W., Michael Hammer, and Daniel E. Thornhill
Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for Use with a Storage-Tube Display Terminal
October 1969 AD 699-930

† TR-64 Rodriguez, Jorge E. (ESL-R-398)
A Graph Model for Parallel Computations, Sc.D. Thesis, EE Dept.
September 1969 AD 697-759

† TR-65 DeRemer, Franklin L.
Practical Translators for LR(k) Languages, Ph.D. Thesis, EE Dept.
October 1969 AD 699-501

TR-66 Beyer, Wendell T.
Recognition of Topological Invariants by Iterative Arrays, Ph.D. Thesis, Math. Dept.
October 1969 AD 699-502

† TR-67 Vanderbilt, Dean H.
Controlled Information Sharing in a Computer Utility, Ph.D. Thesis, EE Dept.
October 1969 AD 699-503

† TR-68 Selwyn, Lee L.
Economies of Scale in Computer Use: Initial Tests and Implications for the Computer Utility, Ph.D. Thesis, Sloan School
June 1970 AD 710-011

† TR-69 Gertz, Jeffrey L.
Hierarchical Associative Memories for Parallel Computation, Ph.D. Thesis, EE Dept.
June 1970 AD 711-091

† TR-70 Fillat, Andrew I. and Leslie A. Kraning
Generalized Organization of Large Data-Bases: A Set-Theoretic Approach to Relations, S.B. & S.M. Thesis, EE Dept.
June 1970 AD 711-060

† TR-71 Fiascanaro, James G.
A Computer-Controlled Graphical Display Processor, S.M. Thesis, EE Dept.
June 1970 AD 710-479

† TR-72 Patil, Suhas S.
Coordination of Asynchronous Events, Ph.D. Thesis, EE Dept.
June 1970 AD 711-763

TR-73 Griffith, Arnold K.
Computer Recognition of Prismatic Solids, Ph.D. Thesis, Math. Dept.
August 1970 AD 712-069

TR-74 Edelberg, Murray
Integral Convex Polyhedra and an Approach to Integralization, Sc.D. Thesis, EE Dept.
August 1970 AD 712-070

TR-75 Hebalkar, Prakash G.
Deadlock-Free Sharing of Resources in Asynchronous Systems, Sc.D. Thesis, EE Dept.
September 1970 AD 713-139

† TR-76 Winston, Patrick H.
Learning Structural Descriptions from Examples, Ph.D. Thesis, EE Dept.
September 1970 AD 713-988

TR-77 Haggerty, Joseph P.
Complexity Measures for Language Recognition, S.M. Thesis, EE Dept.
October 1970 AD 715-134

TR-78 Madnick, Stuart E.
Design Strategies for File Systems, S.M. Thesis, EE Dept. & Sloan School
October 1970 AD 714-269

TR-79 Horn, Berthold K.
Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View, Ph.D. Thesis, EE Dept.
November 1970 AD 717-336

TR-80 Clark, David D., Robert M. Graham, Jerome H. Saltzer and Michael D. Schroeder
The Classroom Information and Computing Service
January 1971 AD 717-857

APPENDIX A

TR-81 Banks, Edwin R.
Information Processing and Transmission in Cellular Automata, Ph.D. Thesis, ME Dept.
January 1971 AD 717-951

TR-82 Krakauer, Lawrence J.
Computer Analysis of Visual Properties of Curved Objects, Ph.D. Thesis, EE Dept.
May 1971 AD 723-647

TR-83 Lewin, Donald E.
In-Process Manufacturing Quality Control, Ph.D. Thesis, Sloan School
January 1971 AD 720-098

TR-84 Winograd, Terry
Procedures as a Representation for Data in A Computer Program for Understanding Natural Language, Ph.D. Thesis, Math Dept.
February 1971 AD 721-399

TR-85 Miller, Perry L.
Automatic Creation of a Code Generator from a Machine Description, EE Degree, EE Dept.
May 1971 AD 724-730

TR-86 Schell, Roger R.
Dynamic Reconfiguration in a Modular Computer System, Ph.D. Thesis, EE Dept.
June 1971 AD 725-859

TR-87 Thomas, Robert H.
A Model for Process Representation and Synthesis, Ph.D., EE Dept.
June 1971 AD 726-049

TR-88 Welch, Terry A.
Bounds on Information Retrieval Efficiency in Static File Structures, Ph.D. Thesis, EE Dept.
June 1971 AD 725-429

TR-89 Owens, Richard C., Jr.
Primary Access Control in Large-Scale Time-Shared Decision Systems, S.M. Thesis, Sloan School
May 1971 AD 728-036

APPENDIX A

TECHNICAL MEMORNANDA†

† TM-10 Jackson, James N.
Interactive Design Coordination for the Building Industry
June 1970 AD 708-400

† TM-11 Ward, Philip W.
Description and Flow Chart of the PDP-7/9 Communication Package
July 1970 AD 711-379

† TM-12 Graham, Robert M.
File Management and Related Topics (Formerly Programming Linguistics Group Memo No. 6, June 12, 1970)
September 1970 AD 712-068

† TM-13 Graham, Robert M.
Use of High Level Languages for Systems Programming (Formerly Programming Linguistics Group Memo No. 2, November 20, 1969)
September 1970 AD 711-965

† TM-14 Vogt, Carla M.
Suspension of Processes in a Multiprocessing Computer System (Based on S.M. Thesis, EE Dept., February 1970)
September 1970 AD 713-989

† TM-15 Zilles, Stephen N.
An Expansion of the Data Structuring Capabilities of PAL
October 1970 AD 720-761

† TM-16 Bruere-Dawson, Gerard
Pseudo-Random Sequences (Based on S.M. Thesis, EE Dept., June 1970)
October 1970 AD 713-852

† TM-17 Goodman, Leonard I.
Complexity Measures for Programming Languages (Based on S.M. Thesis, EE Dept., September 1971)
September 1971 AD 729-011

† TM-18 Replaced by TR-85

† TM-19 Fenichel, Robert R.
A New List-Tracing Algorithm
October 1970 AD 714-522

APPENDIX A

† TM-20 Jones, Thomas L.
A Computer Model of Simple Forms of Learning
January 1971 AD 720-337

† TM-21 Goldstein, Robert C.
The Substantive Use of Computers for Intellectual Activities
April 1971 AD 721-618

† TM-22 Wells, Douglas M.
Transmission of Information Between a Man-Machine Decision System and Its Environment
April 1971 AD 722-837

† TM-23 Strnad, Alois J.
The Relational Approach to the Management of Data Bases
April 1971 AD 721-619

† TM-24 Goldstein, Robert C. and Alois J. Strnad
The MacAIMS Data Management System
April 1971 AD 721-620

† TM-25 Goldstein, Robert C.
Helping People Think
April 1971 AD 721-998

* * * * * * * * * * *

† Project MAC Progress Report I
to July 1964 AD 465-088

Project MAC Progress Report II
July 1964-July 1965 AD 629-494

† Project MAC Progress Report III
July 1965-July 1966 AD 648-346

Project MAC Progress Report IV
July 1966-July 1967 AD 681-342

Project MAC Progress Report V
July 1967-July 1968 AD 687-770

APPENDIX A

Project MAC Progress Report VI July 1968-July 1969	AD 705-434
Project MAC Progress Report VII July 1969-July 1970	AD 732-767

* Copies of all MAC reports listed in Appendix A, as well as earlier Progress Reports, have been deposited with DDC; using the appended AD number, a report may be secured from the National Technical Information Service, Operations Division, Springfield, Virginia, 22151. The prices from NTIS are: microfilm $0.95; hard copies: reports more than two years old $6.00, all others are $3.00 except TR-83 which is also $6.00.

† Out-of-print, may be obtained from NTIS (see above).

†† All TMs have been deposited with DDC and are available only from NTIS, using the AD number appended; the cost is $0.95 for microfilm and $3.00 for hard copy.

ARTIFICIAL INTELLIGENCE

AUTOMATA THEORY

CELLULAR AUTOMATA

COMPUTATION STRUCTURES

COMPUTER SYSTEMS RESEARCH

DYNAMIC MODELING, GRAPHICS AND NETWORKS

EDUCATION

IMPLICIT COMPUTATION

INTERACTIVE MANAGEMENT SYSTEMS

MATHLAB

PROGRAMMING LANGUAGES