N72 25224

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.

Reproduction of this report, in whole or in part, is permitted for any purpose of the United States Government. The distribution of this document is unlimited.





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)

1a. 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	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
1 July 1971	235	(In Text)

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
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	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	12. SPONSORING MILITARY ACTIVITY
None	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 NO. 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM THE BEST COPY FURNISHED US BY THE SPONSORING AGENCY. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE.



ARTIFICIAL INTELLIGENCE

AUTOMATA THEORY

CELLULAR AUTOMATA

PROJECT MAC
PROGRESS REPORT VIII
JULY 1970
JULY 1971

COMPUTATION STRUCTURES

COMPUTER SYSTEMS RESEARCH

DYNAMIC MODELING, GRAPHICS AND NETWORKS

EDUCATION

IMPLICIT COMPUTATION

INTERACTIVE MANAGEMENT SYSTEMS

MATHLAB

PROGRAMMING LANGUAGES

TABLE OF CONTENTS

PERSONNEL		iv
PREFACE		xi
I	ARTIFICIAL INTELLIGENCE	1
	A. Vision and Description	1
	B. Appearance and Illusion	1
	C. Analysis of Visual Scenes	1
	D. Description and Learning	1
	E. Knowledge and Generality	1
II	AUTOMATA THEORY	
	A. Abstract Complexity Theory	3
	B. Algorithms	4
	C. Polynomial Evaluation	4
	D. Sorting	4
	E. Papers	4
III	CELLULAR AUTOMATA	7
IV	COMPUTATION STRUCTURES	11
	A. Introduction	11
	B. Petri Nets	11
	1. State Machines	12
	2. Marked Graphs	12
	3. Free Choice Nets	13
	4. Simple Petri Nets	13
	5. General Petri Nets	13
	C. Asynchronous Speed-Independent Circuits	20
	D. Base Language	26
	E. Program Graphs	30
	F. Translation of Block-Structured Languages	34
	G. Cycles in Structures	46
	H. Computers and People	52
V	COMPUTER SYSTEMS RESEARCH GROUP	57
	A. Introduction	57
	B. Dynamic Reconfiguration	59
	C. I/O Programming Language	60
	D. Automatically Managed Multilevel Memory	60
	E. Protection of Programs and Data	62
	F. System Programming Language	62
	G. Message Handling	63
	H. Graphics Support	64
	I. Other Activities	65

TABLE OF CONTENTS (continued)

J. Acceptance of Multics 66
K. ARPA Network Status 67
1. Design Issues 67
2. Implementation 68
3. Experiments 69

VI DYNAMIC MODELING, COMPUTER GRAPHICS AND COMPUTER NETWORKS 73
A. Introduction 75
B. Dynamic Modeling 76
1. Medistar and the Intravent CA 76
2. The Library of Subroutines 78
3. The Library of Documents 79
4. An Extension of the MIMIC Language 80
5. Lexicontext 81
C. Computer Graphics 82
1. "Picture Framing" 82
2. Polyvision 83
3. Graphical Debugging 83
4. Education 84
5. Visual Statistical Analysis 84
6. Imlac Displays 85
D. Computer Networks 85
1. Network Control Program 85
2. The Network at the End of the Year 86

VII EDUCATION 89

VIII IMPLICIT COMPUTATION 93
A. Introduction 95
B. Exact-Inexact Machines and Approximation 96
C. Pressure-Flow Machines 98
D. Fundamental Work 99

IX INTERACTIVE MANAGEMENT SYSTEMS 103
A. Introduction 105
B. Set-Theoretic Data-Manipulation Systems 105
C. Management Information Systems 106
D. Studies of Access Control and Privacy 107
E. Modeling of Organizations 107
F. SIMPL Project 108

TABLE OF CONTENTS (continued)

X MATHLAB 111

XI PROGRAMMING LANGUAGES 117
A. Introduction 119
B. Canonic Systems 120
C. Power of Canonic Systems 121
D. Canonic Systems and Recursive Sets 70
E. Generalized Translators 124
F. Canonic Reduction Computers 122
G. Undecidability of Problem-Oriented Languages 123
H. Measure Functions of Programming Languages 123
I. Programming Systems Environment 123
J. Community Activities 124
K. Teaching 124

APPENDIX A 221
Project MAC Publications 221

PRECEDING PAGE BLANK-NOT FILMED

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)
Dr. 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. Brodier	Assistant Business Manager (to September 1970)
P. J. Marman	Assistant to the Director
D. G. Draper	Administrative Assistant (to June 1971)
M. K. Buckley	Librarian (to December 1970)
B. M. Kohl	Librarian

Academic Staff

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

Administrators, Research Associates, Research Assistants, Technical Assistants and Visitors

R. G. Abramson	J. E. Gaertner	A. Mathis
V. Alexan	M. Hsiao	E. [illegible]
D. Asthana	M. Hamm	R. M. Mell
J. Bagert	J. F. [illegible]	A. F. Monroe-Davies
E. Barbier	I. I. Hower, [illegible]	M. I. Morse-Gagne
R. H. Bressler	P. G. Neumann	G. Neil
B. Brown	D. A. Henderson	R. L. Ott
G. M. Bruce-Dawson	G. [illegible]	H. F. Ogilvie
K. M. Bryce	F. [illegible]	P. Olson
R. Bryant	J. Johnson	P. C. [illegible]
I. R. Campbell-Grant	R. Johnston	[illegible] Priest
D. D. Clark	M. J. Jaliski	R. T. Rogers
J. Coffman	J. Kaplan	J. H. Rogers
J. D. D'Treville	D. W. Khoury	C. Ramchandani
M. W. Dixon	P. A. King	L. J. Rotenberg
G. T. Dixon	W. [illegible]	J. E. Rumbaugh
R. S. Evans	D. Konig	R. H. Scheil
R. Earle	B. [illegible]ster	M. D. Schroeder
M. Elelberg	J. P. Linderman	J. I. Seiferas
R. D. Feteman	M. P. Lum	A. Sekino
R. J. Fleischer	N. A. Lynch	W. C. Shaw
J. Fosteen	C. W. Lynn	D. G. Sitler
P. J. Fox	R. T. Maddick	J. R. Sloan
M. Fortek	R. Mandl	B. J. Smith
R. C. Goldstein	E. Manning	J. R. Stinger
A. Gonzalez	M. J. Marcus	R. H. Thomas
L. L. Goodman	S. P. Mason	W. M. Toong
I. G. Greif	[illegible] Schoen	M. E. Travis

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

[illegible] Vilfan	P. S. Wang	T. A. Welch
C. M. Vogt	S. A. Ward	C. Wiley
W. C. Walker	A. B. Weathers	D. Yuh

Undergraduate Students

K. F. Bauer	R. M. Elkin	M. V. Koun
A. J. Baum	P. Fralyeigh	D. S. Krackhar
P. B. Bishop	R. A. Freeman	P. B. Kyrk
R. M. Lerma	D. E. Goet	P. [illegible]
E. H. Black	M. R. Genesereth	P. J. Lynch
E. G. Bras	R. S. Goldhor	P. D. Lobling
J. A. Brenfleck	P. J. Green	C. K. Leung
D. Bricklin	R. A. Gurda	J. C. Lind
H. Brodie	P. H. Guldberg	M. Liu
H. L. Brooks	R. H. Gumpertz	W. S. Mark
M. S. Broos	J. B. Harris	J. S. McCarley
K. M. Brown	C. A. Hatvany	V. Minunno
R. H. Brown	B. Hubbard	S. Morrow
S. Carlson	P. W. Hughett	B. G. Morton
A. Y. Chan	W. F. Hui	W. Y. Ng
D. T. Chang	J. E. Jagodnik	P. A. Panunzio
D. M. Christie	K. Kant	G. Pavel
S. S. Cohen	P. A. Karger	R. Pingul
J. R. Cone	R. M. Katz	R. L. Prekken
R. G. Curley	C. A. Kessel	J. Quimby
S. E. Cutler	H. J. Kim	G. S. Reed
B. K. Daniels	R. N. King	J. L. Reuss
R. Davis	E. Kohn	K. Saccades

Undergraduate Students (cont.)

M. A. Rondio	R. P. Silberstein	W. H. Shrader
E. C. Rosen	N. Singer	E. Tsiang
J. D. Rosenberg	A. M. Spilish	H. Tudkor
L. M. Rubin	J. Stori	[illegible]
R. C. Ryan	S. N. [illegible]	[illegible]
J. Saunders	J. M. Schyhorn	[illegible]
M. J. Siegel	[illegible]	

DSR Staff

M. J. Alexiou	S. W. Gaiity	M. A. Padlip
[illegible] Barlin	R. [illegible] Gardner	[illegible]
E. B. Banks	C. C. Garman	W. W. Plumm
A. K. Bhushan	F. P. Goldberg	C. L. Reeve
G. R. S. Bingham	J. P. Golden	L. R. Rothschild
P. Brandler	F. M. Gunkel	D. C. Scanlon
A. L. Brown	R. J. Harman	R. Scarboppel
D. E. Burmaster	J. F. Jarvis, III	F. P. Skinner
B. Byer	R. K. Kanodia	M. J. Spier
R. H. Campbell	R. L. Kohl	A. J. Stifle
H. Q. Capps	M. Lenot	J. Taggart
M. A. Cohen	R. F. Mabee	R. C. Theurer, Jr.
D. G. Crossey	K. J. Martin	A. Varma
R. C. Daley	R. M. Metcalfe	T. L. Woydock
S. O. Dunten	E. W. Meyer, Jr.	B. S. Walker
A. C. England	J. C. McHenry	M. F. Weaver
R. J. Feiertag	R. L. Morris	S. A. Wolfur
J. A. Friel	S. E. Nolan	D. M. Wolf

Support Staff

W. M. Gaffity
R. B. Golden, III
M. K. Hadley
M. A. Hoos
R. Kontrimus
E. T. Moore
R. A. Merrill
F. F. Nanula
G. Pantaleone
R. W. Pierce

Guests

F. Flaemmerg
Prof. J. P. Elkind

J. E. Pinella
R. Pinsley
R. Queen
T. N. Roderick
M. Rubin
H. Schmeer
K. K. Simpson
R. B. Speare
J. Stavrinos
M. K. Stephens
A. G. Teasel
R. B. Ulman
M. W. Webner
L. F. Yapla
P. Yee
R. Young

Prof. M. Fleisher
Prof. G. Jazeolla

PRECEDING PAGE BLANK-NOT FILMED

Administration

Academic Staff

Prof. J. C. T. Pichler
Prof. F. Freed
Prof. R. M. Hedge

Director (to June 30, 1971)
Acting Director (July 1, 1971 to June 30, 1972)
Director (July 1, 1972 to December 31, 1972)

Administrative Staff

C. [illegible] Canton
L. H. Burmeister

Administrative Officer
Assistant Director for Student Activities (to August 31, 1971)
Assistant Director for Plant and Facilities Management (to December 31, 1972)

G. H. Walker
R. Brandler
R. [illegible] Nathan
M. K. [illegible] Frey

Business Manager
Assistant Business Manager (to September 1970)
Assistant to the Director
Administrative Assistant (to June 1971)
Librarian (to December 1970)
Librarian

J. R. Hadley
E. H. Pearl

Research Staff

P. A. Gunnel

Research Assistant

F. Manning

Support Staff

R. [illegible] Scot
L. [illegible] DeCavallaro
M. J. Connell
J. Considine
L. N. Denison
L. L. Gamwell
L. L. Goss
N. [illegible] Greeley
O. Kontrimus

E. T. Moore
G. G. Patanlone
R. W. Pierce
R. Pinsley
E. M. Roderick
K. K. Simpson
R. B. Speare
J. Stavrinos
R. B. Ulman

BEST AVAILABLE COPY

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

In the year ending June, 1970, there were 377 persons associated with MAC. They included: 21 faculty members mainly from the departments of Electrical Engineering and Computer Science and the Alfred P. Sloan School of Management; 106 staff members (RSR Staff and Support Staff); 187 students (Undergraduate and Graduate) and 12 Guests.

Early in its history, MAC conducted a primary research activity with and development of the Compatible Time-Sharing System (CTSS), an early large-access, multiprocessing computer system. More recently we have continued our research in 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 wide and capable facility.

The second of MAC's original objectives, machine-aided cognition, has not only made very significant progress in the last year but the present MAC activities in this area constitute an enormous step forward. In December, 1970 the Artificial Intelligence Group became an independent M.I.T. laboratory. Professors Marvin Minsky and Seymour Papert are continuing as important and key contributors to research in artificial intelligence, primarily as members of the faculty.

In May, 1971, Professor Robert Fano retired from the Directorship of MAC to devote more time to his own research specialties -- Information Science, Computer Graphics, and Computer-Networks. Professor M. Frankin assumed the Directorship. Miss Jerrine Pearson 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 research groups. Educational Applications (MacAIMS: Programming Linguistics/Extensible Languages and Programming Linguistics/Formal [illegible] have been established 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 problem of embedding knowledge in the computer and in enabling that knowledge-capable system to play a key role in generating programs and other forms of solutions to problems. We believe that by so doing we can make systems able to play an active cooperative role with its users. We [illegible] this field the name "Artificial Intelligence".

Programming

This project began [illegible] the year ending [illegible] MAC.

Professor Robert Fano

Professor Marvin Minsky

Professor Seymour Papert

Computer Science Laboratory

Massachusetts Institute of Technology

Cambridge, Massachusetts 02139

This research is supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0001.

Additional support is provided by the National Science Foundation under Grant Number GJ-34671.

AUTOMATA THEORY

Prof. [illegible]

Academic Staff

Prof. [illegible]
Prof. A. Meyer

Instructors, Research Associates, Graduate Assistants and Others

A. Bayer
W. Brown
G. G. [illegible]
M. Fellows
M. M. Hammer
T. N. Hutchins
D. J. Moury

Support Staff

M. [illegible]
V. M. [illegible]
S. T. [illegible]

II. AUTOMATA THEORY

Abstract complexity theory, which has been a central topic of research in the Automata Theory group for the past few years, has now reasonably developed chapter in the theory of computability, with contributions from nearly three dozen authors in the US and the Soviet Union. As reported below, some further progress in this area was obtained during this last year, and new developments are still in progress. However, the topic which has associated with the classification of computations according to the time and space requirements they place on them, has understood and further refinements in the direction they appear to be dimensionless interest to theoretical computer scientists. Major interest within the group has been directed 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 develop techniques for verifying the optimality of algorithms. The work of Arlazarov, Trakhtenbrot and Winograd on matrix multiplication, polynomial evaluation, and parsing 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 computably functions f, there exists a zero-one valued computable function g that takes more than time t(x) to compute. More precisely, any program that computes g requires at least t(x) steps to compute g(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 g are easy to compute. It might be the case that the functions g that are difficult from the point of view of complexity theory are the functions that are time-consuming to compute on the average - are actually easy for all small arguments, i.e., for all arguments less than [illegible]. In fact any zero-one valued function can be computed quickly for any given finite set of arguments by simply storing in a table the correct values of the function in question. A [illegible] complex function should have the property that any program that computes it can run rapidly on only so 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 [illegible]).

[illegible] of program size complexity 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 conciseness of different programming languages for computing partial functions. The size of the program needed to implement the [illegible] in different languages. A fairly general theorem of this type, proved by [illegible], shows that a sufficiently [illegible] programming language [illegible] a program size [illegible] requires [illegible] by any [illegible] language.

M. J. Fischer derives quantitative bounds on the improvement in simplicity of description that can be achieved by using [illegible] grammars such as context-free grammars [illegible] machine [illegible] to data [illegible] even [illegible]

c. Algorithms on Graphs

The [illegible] matrix multiplication and transitive closure of graphs [illegible] last year's report have been strengthened. Robert Tarjan 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 n-by-n boolean matrices. When combined with Meyer and Fischer's observation that fast algebraic methods for multiplying polynomials can be modified to apply to boolean matrices, this result yields the best transitive closure algorithm known to date.

Our hope that a [illegible] boolean matrix multiplication might enable us to generalize fast [illegible] multiplication techniques has not yet been fulfilled. We continue to believe that this approach is promising.

d. Polynomial Evaluation

The evaluation of rational functions by sequences of [illegible] operations represents one of the few areas where techniques have been developed for establishing the optimality of algorithms. Larry Stockmeyer, together with Professor Albert Meyer and M. S. Paterson, has derived [illegible] bound on the number of multiplications required to evaluate the determinant of [illegible] with rational coefficients, and has shown that this lower bound is nearly attainable.

e. Sorting

[illegible] has investigated [illegible] networks composed of two-input, two-output comparator elements. Each comparator element of a three-wire network sorts three numbers, the resulting network is a machine that sorts any sequence of numbers presented to it. When implemented in hardware, such networks can be viewed as high-speed sorting devices. Although [illegible] sorting of data is performed by a computer program that is naturally suited to parallel evaluation.

Smith has been studying the number of comparator stages required to construct an n-line sorting network. He has discovered that a network of comparators actually sorts if and only if the network can

$$\sum_{i=1}^{n} \binom{n}{i} = 2^n - 1$$

[illegible]

PRECEDING PAGE BLANK-NOT FILMED

CELLULAR AUTOMATA

Stephen Wolfram

IAS Staff

Member

Undergraduate Visitors

Name

PRECEDING PAGE BLANK-NOT FILMED

CELLULAR AUTOMATA

PRECEDING PAGE BLANK NOT FILMED

COMPUTATION STRUCTURES

Prof. J. B. Dennis

Academic Staff

Prof. R. H. Kane — Prof. [illegible]

Instructors, Research Associates, Research Assistants and Others

I. R. Campbell-Grant	[illegible] Lester
D. Coffman	[illegible] Linderman
J. Fosbean	H. J. [illegible]
F. J. Fox	C. [illegible]
F. Furtek	[illegible] Rosenberg
I. G. Grief	[illegible]
M. Hack	[illegible] Stiffer
T. Hawryszkiewycz	W. C. Walker
P. G. Hebalkar	

Undergraduate Students

H. J. Kim

DSR Staff

W. W. Plummer

Support Staff

B. A. [illegible] — A. Rubin

1

PRECEDING PAGE BLANK NOT FILMED

I. COMPUTATION STRUCTURES

A. Introduction

The Computation Structures Group is concerned with the study and analysis of fundamental [illegible] in the design and construction of general-purpose computer [illegible] encompasses hardware and software [illegible] systems, and much of the work [illegible] a common conceptual [illegible] for both [illegible] one of the theoretical [illegible] asynchronous systems of [illegible] Petri nets [illegible] systems. The goal of this work [illegible] to serve as the basis of a new methodology for the design of asynchronous digital systems. The second area [illegible] a base program language. [illegible] formal definition scheme for [illegible] programming languages and will provide a sound [illegible] 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 [illegible] and for studying asynchronous systems [illegible] in relation to other [illegible] and with [illegible] parallel [illegible] solution [illegible] have the simple [illegible] Petri nets [illegible] study of subclasses of Petri nets [illegible] toward understanding the general [illegible] has been an important objective [illegible] past year. We have identified several [illegible] and have found useful results about them. Before discussing these results, we present a brief introduction to Petri nets and the subclasses of interest.

[illegible]

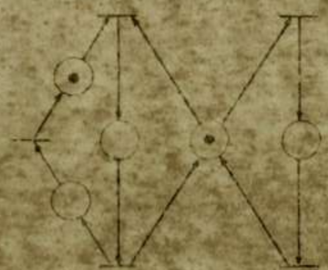


FIG. 1 A PETRI NET.

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 and [illegible] 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 (SPN)

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 transitions and places are permitted for each class of nets.

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



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



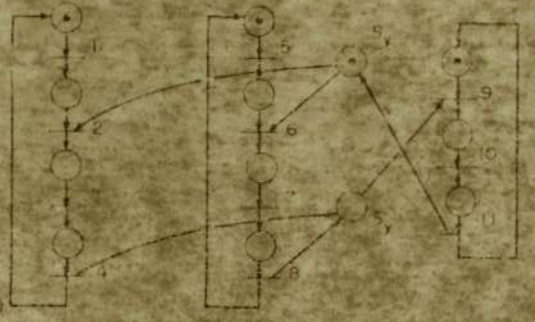


PROCESS

```
P1                      P2                      P3
1  x ← x+1              5  z ← z+y              9  P(Sy)
2  P(Sx)                6  P(Sx)                10 z ← z-y
3  y ← y+1              7  y ← y-1              11 V(Sy)
4  V(Sy)                8  V(Sx)                GOTO 9
   GOTO 1                  GOTO 6
```

INITIAL SEMAPHORE $S_x = 1$ AND $S_y = 0$

a)



b)

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

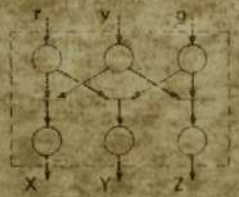


FIG. 7 THE 2-OUT-OF-3 NET.



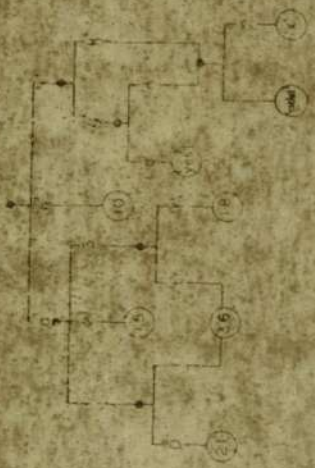


FIG. 6.

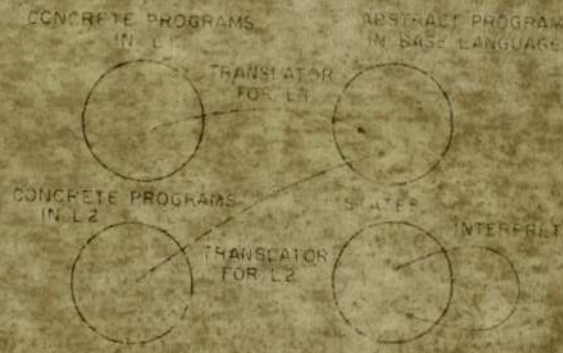

CONCRETE PROGRAMS
IN L1
TRANSLATOR
FOR L1
ABSTRACT PROGRAMS
IN BASE LANGUAGE
CONCRETE PROGRAMS
IN L2
TRANSLATOR
FOR L2
INTERPRETER

FIG. 9

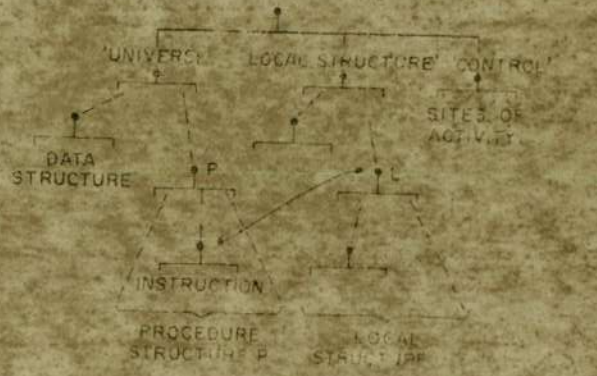

UNIVERSE
LOCAL STRUCTURE
CONTROL
SITES OF
ACTIVITY
DATA
STRUCTURE
P
L
INSTRUCTION
PROCEDURE
STRUCTURE P
LOCAL
STRUCTURE

FIG. 10



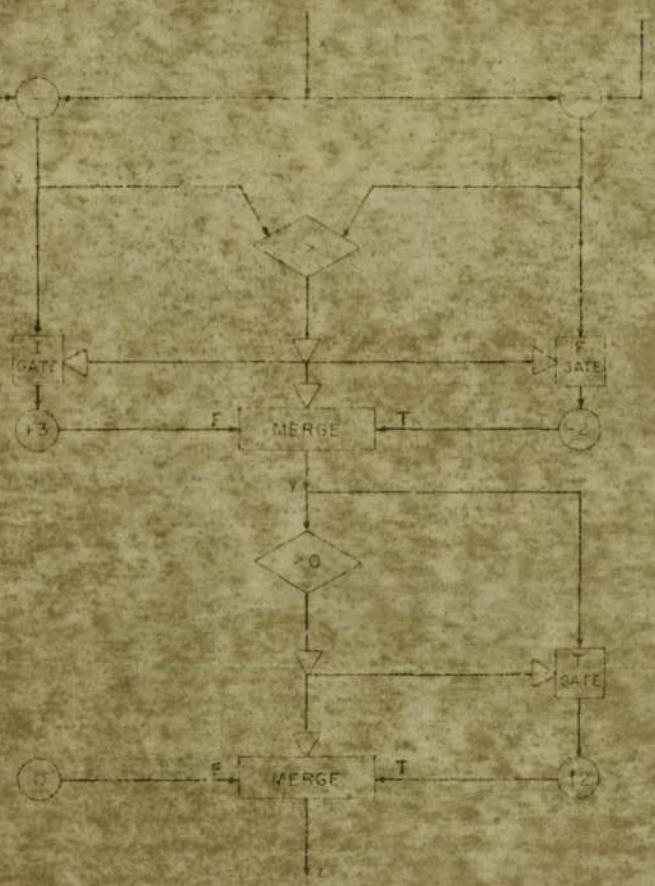


FIG 11

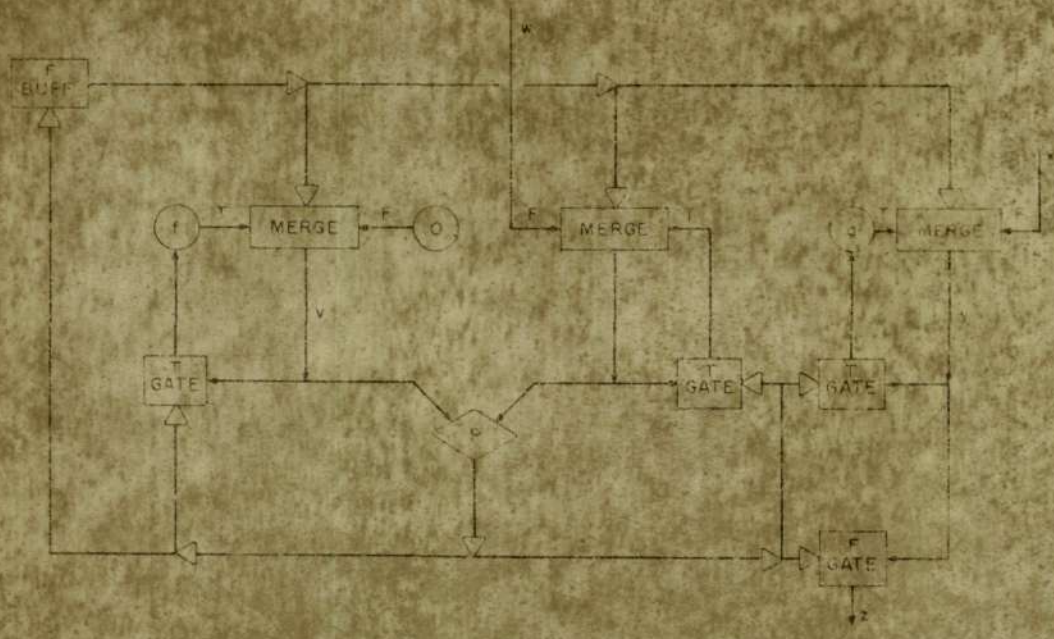


FIG. 4.

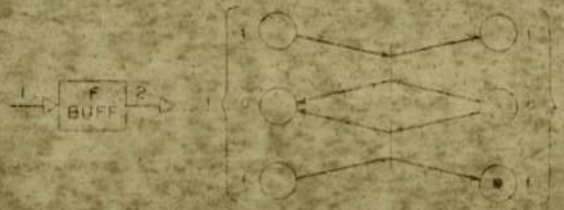


FIG. 15.

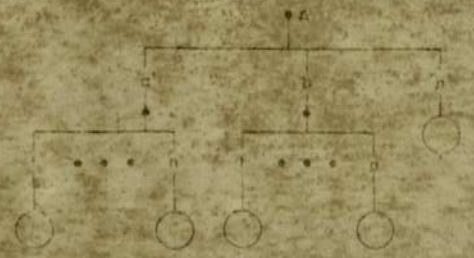


FIG. 16.

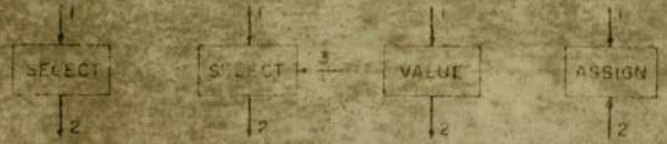


FIG. 17.

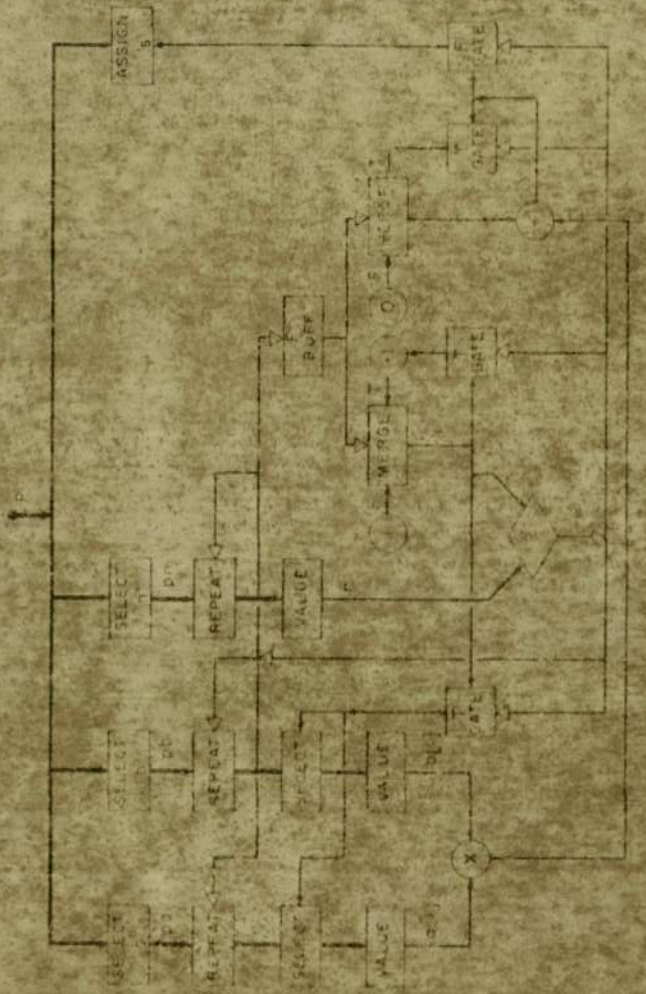
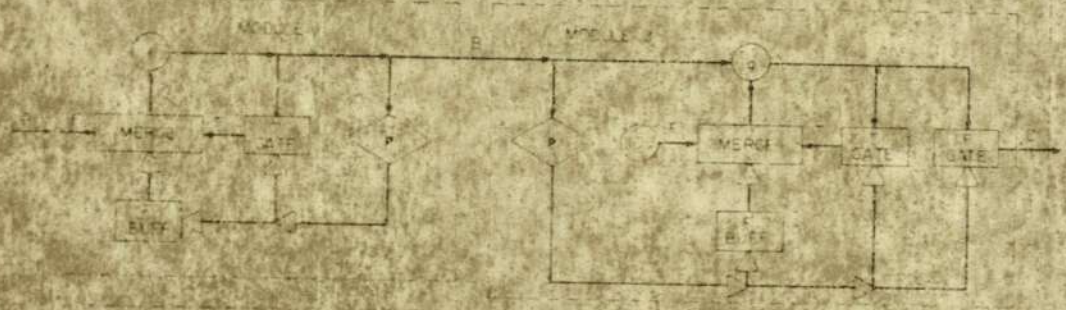
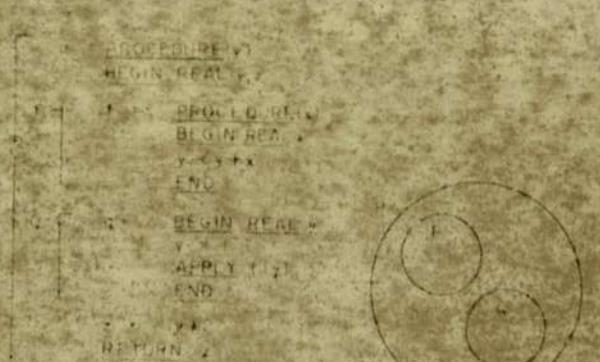


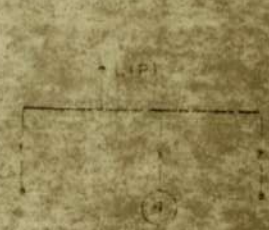


FIG. 19




PROCEDURE P;
BEGIN REAL x;
PROCEDURE Q;
BEGIN REAL x;
x:=3.14;
END;
BEGIN REAL x;
x:=2.71;
APPLY Q;
END;
x:=1.41;
RETURN;
END;


P
TEXT Q
TEXT P


P

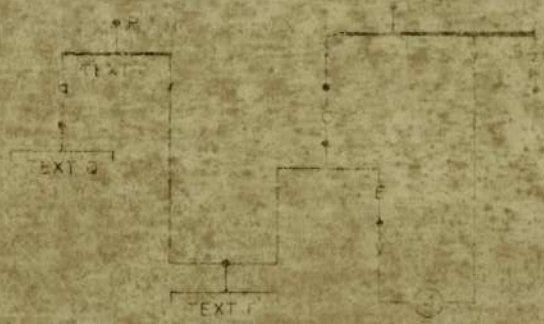


FIG 23.

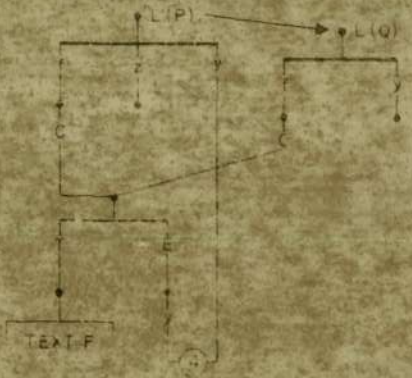


FIG 24.

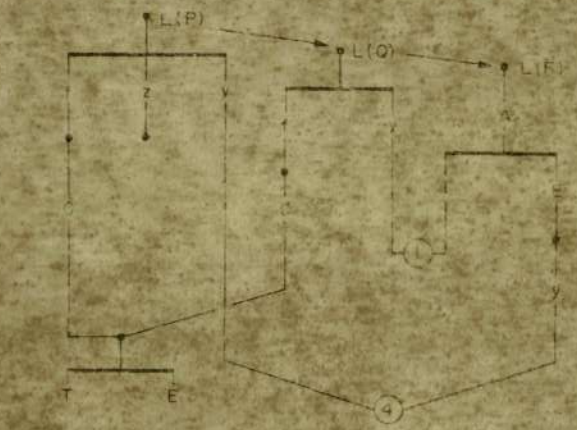


FIG. 25.

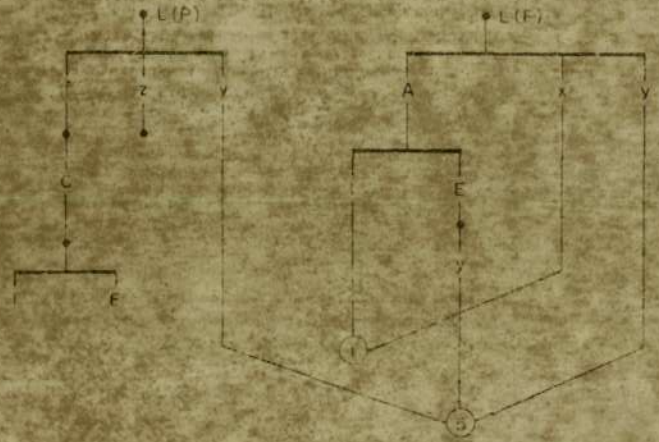


FIG. 26.


```
P := PROCEDURE();
BEGIN
    Q := PROCEDURE(x); INTEGER x;
    BEGIN
        IF x = 0 THEN RETURN 1;
        x := Q(x);
        z := APPLY F(x);
        RETURN z;
    END
    APPLY F(6);
END
```

FIG. 27.

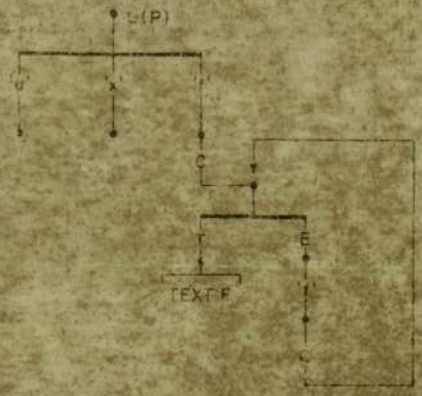


FIG 28.

2 OF 4
N72 25224 UNCLAS
1.0
1.1
1.25 1.4 1.6

```
P ┌ p := PROCEDURE(u)
  │ BEGIN
  │   ┌ f := PROCEDURE(k,x) PROCED n; INTEGER x
  │   │ BEGIN
  │ F │   IF x = 0 THEN RETURN
  │   │   x := g(x)
  │   │   z := APPLY f(k,x)
  │   │   RETURN z
  │   │ END
  │   └ APPLY f(f,u)
  │ END
  └
```

FIG. 29.

Computers and People

When computers are used in the service of society, the specific characteristics of the computer system employed critically influence the behavior of the larger system containing it. Hardware, software and people have all learned by now that some 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 once effective, controllable and tolerable to human standards.

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, 1975, at the Stevens Institute of Technology.

Work in the second area consists mainly of doctoral research by Jerome H. Saltzer. 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 spheres to spheres through inter-sphere links. It can be shown that, under appropriate conditions, call-line spheres cannot spy on their callers, or trap-a-level on their callers. The model includes facilities for keeping records of critical actions by spheres of programs, for instance, and for allocating responsibility for whatever a program does. Such facilities are essential to implement and enforce legislations and contractual agreements existing in the user community. A type of summary description of this work is presented in one of the papers listed below. Certain surveillance mechanisms in a secure computer facility

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 (June, 1970), pp. 3-52.

2. C. A. Petri, Communication with Automata, Supplement 1 Technical Report RADC-TR-65-377, Vol. 1, Rome Air Force Base, New York, 1966. (First roughly published in German under the title Kommunikation mit Automaten, University of Bonn, 1962.)

3. E. W. Dijkstra, Cooperating Sequential Processes, Technische Hogeschool Eindhoven, Netherlands, 1965. Reprinted in Programming Languages, F. Genuys, Ed., Academic Press, New York, 1968.

4. A. W. Holt and F. Commoner, Events and Conditions, Rept. 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, Cooperating 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. [illegible] B. Dennis and J. B. Fossum, Modularity in 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, Progress in Program Generality, Parallelism in Computer Systems, Information Processing 68, North-Holland, Amsterdam 1969, pp. 484-492.

11. J. B. Dennis, Future Trends in Time Sharing Systems, Proceedings International Symposium and Summer Institute for Operations Research and Econometrics, Systems Operations Research Council, 1967, pp. 222-235.

12. [illegible] Computational Structures Group Memo 78, Project MAC, M.I.T., Cambridge, Mass., June 1971.

13. [illegible] Recent Results on the Formal Complexity [illegible] Proceedings of the [illegible] Conference on [illegible] Computer Science, [illegible] 1971.

14. [illegible] A Type of Computation [illegible] Programming Languages and Computer Programming, [illegible] Amsterdam, 1972, pp. 1-12.

References (cont.)

15. [illegible] Liddle, The Mechanical Evaluation of Program Schemas, The Computer Journal, Volume [illegible], Number [illegible], January 1970, pp. 308-313.

16. J. B. Dennis, A Graph Model for Parallel Computation, Technical Report MAC-TR-64, Project MAC, M.I.T., Cambridge, Mass., September 1967.

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

18. [illegible] Petri, Concurrency and Parallel Computation, Princeton Conference on Information Sciences and Systems, Princeton, N.J., March 1971.

19. [illegible] B. Dennis, On the Design and Specification of a Common Base Language, Proceedings of a Symposium on Computers and Automata, Polytechnic Institute of Brooklyn, Interscience Publishers.

20. M. Campbell Grant, "The Controlled Execution of Parallel Programs," Memorandum [illegible] Project MAC, M.I.T., Mass., Dept. of Electrical Engineering, January 1971.

Publications 1970-1971

Campbell Grant, M., "The Controlled Execution of Parallel Programs," Memorandum [illegible] Project MAC, M.I.T., Mass., Dept. of Electrical Engineering, January 1971.

Dennis, J. B., "Coroutines and Parallel Computation," Proceedings of 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, [illegible].

Dennis, J. B., and Patil, S. [illegible] Asynchronous Circuits, in Proceedings of the Sixth Annual Princeton Conference on Information Sciences and Systems, [illegible].

Patil, S. S., "Coordination of Asynchronous Events," Project MAC Progress Report, March 1971, pp. 9-13.

Publications (cont.)

Lang, R. M., "Computers in Society", to be published in the Proceedings of the Symposium "Informatics, 10 Years Later", Societa Italiana, held at the Fondazione Cini in San Giorgio, Venice, Italy, October 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 59, Project MAC, M.I.T., Cambridge, Mass., February 1971.

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

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

PRECEDING PAGE BLANK NOT FILMED

COMPUTER SYSTEMS RESEARCH

Prof. F. J. Corbató

Academic Staff

Prof. J. H. Saltzer

Instructors, Research Associates, Research Assistants and Others

D. D. Clark	P. F. Bendik
J. Coffman	M. D. Schroeder
F. T. Pogran	A. Sekino

Undergraduate Students

T. B. Bishop	N. S. Lincoln
D. Brickl[illegible]	M. Lin
D. Carlson	D. Minunas
J. W. Cone	D. F. Koch
D. Frank	R. Rhodes
[illegible] A. Green	J. Stein
W. [illegible] Gumpert	J. M. Strathorn
[illegible] A. Harper	C. D. Tavar[illegible]

DSR Staff

[illegible] Gumpert	E. W. Meyers, Jr.
R. [illegible] Daley	N. I. Morris
S. R. Dalton	M. A. Padlipsky
M. [illegible] Feiertag	I. D. Skinner
R. L. Gardner	M. J. Spier
C. C. Garman	V. L. Voydock
R. [illegible] Ghodsi	N. [illegible] Weaver
F. F. Mah[illegible]	[illegible] Webber

Support Staff

[illegible] Carey	D. E. Jones
[illegible]	T. H. Seymour
C. F. [illegible]	A. G. Testa
[illegible] Barr	R. [illegible] Wer[illegible]

Guest

Prof. G. Iazeolla

PRECEDING PAGE BLANK-NOT FILMED

V. COMPUTER SYSTEMS RESEARCH

A. Introduction

The Computer Systems Research Group concentrates upon discovering and developing the fundamental principles of computer information systems. Its goal is to provide the basis for the design of computer systems which are more powerful, more flexible, and more reliable than those now available. In contrast to the more theoretical areas of computer science, the Computer Systems Research Group is concerned with the practical problems of designing and implementing actual computer systems. Such work has both advantages and difficulties. The chief advantage is contact with reality and the knowledge that the results of research will be applied in real operating environments. The chief difficulty is the need to build and maintain large software systems which is very costly and requires considerable expertise. The chief drawback is the one mentioned above: the pressure to produce results which will be applied in operating environments leads to a research group that is dominated by short-term projects, not all of which are worthwhile.

Taken in proper balance, these two considerations can lead to the use of a live system as a laboratory in which a number of good research problems can be adequately attacked, but careful planning is required. 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
- PL/I Programming Language
- Message Handling

Each of these areas will be discussed in turn.

B. Dynamic Reconfiguration

As the "computer utility" is ever increasing its number of users, the reliability of the electric power utility or the telephone company, upon which it depends, is approaching that of the individual components which are fairly reliable. Therefore, a computer utility must have the ability to provide service to its users despite the failure of certain components. Ideally, the configuration of the computer should be dynamically modified to the changing demand, with one machine handling the load during peak periods, while another is taken down for maintenance. Ideally, this should be accomplished without any disruption of service to the users. Although this goal has long been proposed for multiprocessor systems, relatively few operating systems, of which the Multics system is one, have been designed to allow the dynamic reconfiguration of the system. Even in the Multics system, the ability to reconfigure dynamically is severely limited. The goal of the research project in this area is to provide the basis for a system which can be dynamically reconfigured in response to changes in the demand placed upon it and in the number of components which are operating correctly.

The first step in this research was the design of a model for describing the dynamic reconfiguration of a computer system. This model was developed by Professor Jerry Saltzer of the Electrical Engineering Department and Professor Jack Dennis of the Electrical Engineering and Computer Science Department. Using this model as a basis, Professor Saltzer has proposed a specific implementation of the model for the Multics system. This implementation will require changes to the hardware of the system, as well as additions to the operating system software. The changes to the hardware will provide the ability to halt and restart individual processors and memories, to change the descriptions of the memory and processor configurations which are used by the hardware in addressing memory, and to cause interrupts to occur when various types of hardware errors are detected. The changes to the operating system software will provide the ability to reconfigure the system in response to commands from the operator or from various programs in the system, and to automatically reconfigure the system in response to the detection of hardware errors. Implementation of the proposed changes is currently underway.

The final phase of this piece of research should be the study of problems beyond the Multics system. Since the thesis provides a general model for such operations, it can provide the designer of a new system with the insight needed to allow him to include dynamic reconfiguration in his computing 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 indeed. 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 strings 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 teletype, IBM 1050 and 2741 terminals, and the Datel 30 and Datel 300 terminals. So far, the new control package has proven much more reliable than earlier designs, thus providing some basis for continued experimentation with I/O channel control languages.

D. Automatically Managed Multilevel Memory

191, and the demand paging virtual memories of Multics, IBM's CP-67, and the Michigan Terminal System. Fortunately, behind the best of these pioneer efforts a wealth of knowledge about how to engineer a multi-level memory exists with appropriate algorithms which are matched to the load and hardware characteristics. One of the projects of the Computer Systems Research Group is to implement and experiment with the Multics memory hierarchy in order to learn better how to predict in advance the performance of proposed, new, automatically-managed, multi-level memory systems. Several specific aspects of this goal have been explored recently:

1. A strategy to front-end a drum, and place it in 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 drum page 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 "misses" made in the virtual memory) suffices to provide a quite accurate prediction of paging and I/O overheads. A doctoral thesis is in progress on this topic.

5. Hardware and programming techniques for [illegible] led to the discovery of an error [illegible] algorithm and a second hardware [illegible]

ingenious algorithms [illegible] two candidates [illegible] hardware [illegible] twice at once [illegible] cannot occur [illegible] for the first [illegible] the two [illegible] thereby processed out of order, performance is [illegible] service is reduced.

[illegible] of Programs and [illegible]

A long-standing objective of the CSR group has been to provide facilities for and protection between programs from one another, so that users of a public computer utility can have confidence, place appropriate portions of their programs and private information in the machine, and be protected from malfunction resulting from other users' programs. In [illegible] provided a general system [illegible] protection relationships. This scheme [illegible] provides user-written subsystems with the same protection for other users that the hardware and supervisor [illegible] the user-written subsystem to be [illegible] This scheme was brought under intensive [illegible] with two results:

1. A hardware architecture which implements the [illegible] proposed. One of the outstanding [illegible] feature is that subroutines can be [illegible] another but exactly the same mechanism [illegible] among procedures within a protected [illegible] proof sufficiently promising that it is included in the specifications for the next generation of hardware to be used for Multics.

2. As an experimental [illegible] feasibility of multi-layered supervisors, several supervisor [illegible] such layers, but not all [illegible] line of protection intermediate between the user and the [illegible] security. The success of this [illegible] such layers [illegible] a practical way of [illegible] behavior code which must be given a privileged [illegible]

Both of these results are [illegible] complete solution to the understanding of [illegible] and later, a [illegible] concept of protection [illegible] tion. Two doctoral [illegible] are underway in this area.

[illegible] Programming Languages

[illegible]

[illegible]

3. Research in techniques of compiling continued, with a major result being a description of the code generation phase which is sufficient to allow the [illegible] 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 [illegible] depend upon problems, 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 these interprocess messages requires a protected capability not [illegible] terms of access to addressable memory. Thus the function of providing protected mailboxes for message queues to be a primitive one which must be provided by the supervisor or the hardware. Although message queues [illegible] have been proposed and implemented in other systems, this is the [illegible] the architecture in a natural way. It is hoped that developed message queues fit into the [illegible] 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 mechanism may be an appropriate object for direct hardware implementation. To explore this area, a software message queue mechanism was designed, and [illegible] and the various independent [illegible] are being developed.

H. Graphics Support

The CSR Group does not carry out research [illegible] graphics display. However, there are many very interesting and important sites in the field of graphics investigations elsewhere, which have not received adequate publicity because they were implemented within the general 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 having sophisticated graphical display [illegible] of the general purpose [illegible] and, several new [illegible] in progress.

[illegible]

[redacted]



PRECEDING PAGE BLANK-NOT FILMED

PRECEDING PAGE BLANK NOT FILMED

VII. DYNAMIC MODELING, COMPUTER GRAPHICS, AND COMPUTER NETWORKS

A. Introduction

[illegible] Group, the Computer Graphics [illegible] [illegible] Group [illegible] Project MAC [illegible] [illegible] of these [illegible] [illegible] interrelated [illegible] computer system [illegible] interactive [illegible] [illegible] Annual Report [illegible] [illegible] computer [illegible] [illegible] members of the [illegible] Research Group, [illegible] reported upon in that group's [illegible] Annual Report.

The [illegible] objective of the [illegible] research [illegible] [illegible] that will [illegible] 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 [illegible], 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 [illegible], and good time-sharing systems [illegible] to some extent facilitate their preparation, but existing systems [illegible] dimensions. They do not provide [illegible] out of which to assemble models. They [illegible] 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 [illegible] of the sets of tools, techniques, [illegible], and models [illegible] [illegible]

[illegible]

MODELING, GRAPHICS, AND NETWORKS

The Dynamic Modeling [illegible] [illegible] [illegible] [illegible] Computer Networks Group [illegible] [illegible] [illegible] [illegible] models.

The three groups share the work of [illegible] the hardware computer facility and adapting the [illegible] system programs to meet [illegible] [illegible] contribute [illegible] directly and [illegible] and understanding of modeling, graphics, and networks, but [illegible] is essential [illegible] innovative work [illegible] [illegible] organization for highly interactive and predominantly graphic man-computer interaction.

[illegible] three basic themes [illegible] involved in the joint [illegible] it has been difficult to find an expressive name for the Dynamic Modeling/Computer Graphics/Computer Networks (DM/CG/CN) system. Despite experimentation with permutations of "dynamic", "graph", "comp", "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 [illegible] 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 [illegible] and data sets, the library of [illegible], [illegible] language based on LISP but extended [illegible] types and to explicit graphics, and a word [illegible] representing text.

1. Mediation and Intervention

"Mediation", as the term is used here, is the function [illegible] that interposes itself [illegible] their intercommunication, between [illegible] routines and the sets of data [illegible] operator of the computer [illegible] collection of [illegible] called "CARE" ("Call and Return") prepared by [illegible], Paul Hugnett, and J. C. R. Licklider. CARE is [illegible] within each process in the system. CARE [illegible]

sequence format will be available [illegible] the [illegible] certain system associated [illegible] and other constant [illegible] cumulate about the software [illegible] able in a small [illegible] and-paper [illegible] of [illegible] cated earlier, we set up a [illegible] version [illegible] on MULTICS last year. [illegible] are beginning [illegible] the PDP-10 system. [illegible] tial part of the concept of the system [illegible]

4. An Extension of the LISP Language

Our design objective of high [illegible] interactive experimental models is inconsistent with the classical organization of software based on compiling and loading a monolithic machine language program in which all the parts are totally linked together. Limitations of the classical organization [illegible] two main directions, on the one hand in the [illegible] in which linking is deferred until [illegible] during the execution of a program [illegible] should be linked to the parts already [illegible] hand in the paradigm of LISP, in which [illegible] other such activities are carried on [illegible] work of the language implementation [illegible] MULTICS and LISP have many desirable features, taken [illegible] and of itself provides the desired [illegible] modeling to which we aspire. To mention [illegible] MULTICS is not set up for use with a [illegible] operating out of main memory, and LISP, while highly [illegible] ally, is difficult to bring into relation with external software and is only weakly developed in the directions of data typing 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 Major Minter of Lynamic Modeling, Computer Sciences. 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 [illegible] with the system's subroutine library.

We hope to use MUDDLE as an interpreter for the upper [illegible] the subroutine-call hierarchy. The idea is to [illegible] for the upper levels, where most of the changes are [illegible]

in the course of exploring a model, and to employ assembled or [illegible] subroutines at the lower levels, where the [illegible] of execution (as opposed to interpretation) is most important, [illegible] such an organization of software may be feasible. On the other hand, it would not be [illegible] version of [illegible] Artificial Intelligence Laboratory because [illegible] and [illegible] seem 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 [illegible] 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.

[illegible] a vocabulary of [illegible] over 32,000 lexical words.

[illegible] and NLS, 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-coded 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.

6. 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 [illegible] channel and shares memory with them. Since memory is dynamically allocated by the operating system to the several user programs that are running concurrently in the PDP-10, it is necessary for the operating system to [illegible] the [illegible] of the display processor [illegible] allocation [illegible]. This has not been a difficult problem so far because we have been using only one E&S display. We have followed the [illegible] 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 timesharing system.

1. "Picture Framing"

[illegible]

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 program designed and programmed by James Michener, Edward Sayre, and others, that permits the modeler to assign the various aspects of his model, as they appear on the display, to areas and then to move the areas about, magnify, name and save them, or 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 modulator CARMA in due course, but it will not be necessary for the modeler to halt the execution of the model to adjust the configuration of his display. The modeler can control the display subsystem while the model is running. This is so done 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 in 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 ahead and then used with various models; the former seems, in the present state of the art, to require model-specific operations.

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 SCOT, 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 computer console, one can open and look inside the memory to see what is there. With a graphic display, on the other hand, one can see a map of the computer memory (either of the internal core memory registers

or in the symbolic space of source program statements) which shows the behavior of the program as it executes.

During the past year, we have done a variety of things to exploit meaningfully graphical display. Bert Sutherland, working with Ivan Sutherland, completed a graphics display system called Lamp, and Paul Huffert completed a set of operations for manipulating tool, called Curt. In both cases, the registers and segments of program memory are displayed upon the display surface, and the flow of information is represented by moving spots. Both Lamp and Curt are powerful tools for understanding program dynamics, but they will, themselves, find useful application in the computer system.

4. Plot Package

The difficulty of harnessing a powerful display subsystem in the interest of man-computer communication about complex problems is balanced by the simplicity of getting such a system to display mathematical functions. It is easy to produce all kinds of "prime pictures" on the display screen, and it is easy to create families of curves and surfaces. It is remarkable how much can be learned from a few minutes of playing 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 past several months of the Laboratory development, graphics programs that provide insight into simple mathematical phenomena. These included two-dimensional and three-dimensional function plotters (Edward Blake, Scott Snyder), a Fourier transformer (Robert Siegel) and a simulation of the interplay of gravitational forces in a galaxy (Paul Huffert).

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 visual routines without the burden of writing both interactive and

10. Imlac Displays

Although not as elaborate [illegible] as the [illegible] Imlac consoles, which [illegible] [illegible] community [illegible] [illegible] [illegible], are [illegible] very capable. [illegible] has, thus far, been largely limited to [illegible] [illegible] display, but we [illegible] to [illegible] for display of graphs, charts, and diagrams.

David Epling prepared a PDP-10 assembler [illegible] Stephen [illegible] prepared a loader that [illegible] PDP-10. Lawrence [illegible] and Stephen [illegible] [illegible] techniques for Imlac consoles [illegible] programs [illegible] to make the Imlac's control [illegible] convenient [illegible] in our applications. And [illegible] designed and implemented a program, IMEDIT, that makes it convenient to use the Imlac editor in conjunction with the PDP-10 time-sharing system. IMEDIT moves [illegible] from the PDP-10 to the Imlac consoles such [illegible] [illegible] ing but trivial work. [illegible] in all, the Imlac consoles are quite convenient and effective for applications that are limited to alphanumeric information, and they [illegible] well on their way to supporting applications [illegible] [illegible] line drawings, also.

B. 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 [illegible] operations mainly in any one of several other ARPA [illegible] machines. Most of the work to be reported [illegible] aimed at creating the basis in computer communication [illegible] programs in the PDP-10, for interaction [illegible] PDP-10 and other network computers.

1. Network Control Program

Robert Bressler and other members of the [illegible] [illegible] developed [illegible] [illegible] [illegible] Control Program (NCP) for the PDP-10 computer. This program [illegible] [illegible] [illegible] [illegible] in the PDP-10 [illegible] ARPANET computers [illegible] [illegible] and implement programs that [illegible] make the PDP-10 a part of an alphanumeric telecommunications network [illegible] within the ARPANET. The TELNET programs are of two kinds, "server" and "user". The first TELNET server program [illegible] was [illegible] the function of which is to permit users at other ARPANET computers [illegible] connected to the [illegible] [illegible] local users. [illegible] of the [illegible] [illegible] Robert [illegible] [illegible] Metcalfe, and [illegible] [illegible] of the programming was [illegible] [illegible].

The next program in the logical hierarchy of network software was a TELNET user program, designed and [illegible] by Robert Metcalfe. Its function [illegible] [illegible] [illegible] the PDP-10 to [illegible] another computer [illegible] [illegible] [illegible] communications [illegible] from the PDP-10, [illegible] communicating with a remote TELNET server program, through the Network Control Program. Together, the TELNET server and user [illegible] [illegible] the [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

Even before the basic means of communication [illegible] [illegible], attention turned toward the [illegible] [illegible] [illegible] network software. [illegible] [illegible] [illegible] for the transfer of data and for the transfer of files of data. His interest led him to the [illegible] of the [illegible] Data and File Transfer Protocol for the ARPANET. [illegible] the reporting year, he was working [illegible] [illegible] of the tentative protocols that [illegible] [illegible]. Members of the Computer Networks [illegible] [illegible] [illegible] [illegible] their attention, mainly, to problems of [illegible] communication through the network.

Rather early in the year, interesting [illegible] of graphical communication through the network were [illegible] in cooperation with members of the Aiken Computation Laboratory at Harvard. Graphic programs were [illegible] at the Harvard PDP-10, transmitted through the network to the Project MAC PDP-10, processed there by the PAS subsystem, transmitted back to a PDP-10 computer at Harvard, and displayed there on a Digital Equipment [illegible] display. The same pictures were displayed on the [illegible] 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 [illegible] continuous motion, whereas the network display [illegible] 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 [illegible] lines, to handle [illegible] graphics.

3. 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 [illegible] [illegible] subsystems and pressing [illegible] into [illegible] [illegible] of the hierarchy of network software. The [illegible] [illegible] consolidation will be time-consuming [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] documentation.

At the same time, there is a keen sense of anticipation within the group, a strong motivation to master the transfer of data [illegible] and to proceed as rapidly as possible to the execution of [illegible] computers of supercomputer capability programs [illegible] to the Applied Research Subprogram Linkage [illegible] for [illegible] within the scope of our Integrated Computer Network. Collections of programs this [illegible] computer [illegible] that is needed and the [illegible] profitable [illegible] program for the [illegible]

References

1. Carnahan, Brice, Luther Rubin, Theodore H. Nelson, David Rice and Andries van Dam, A Hypertext Editing System for the 360, Proceedings Conference on Computer Graphics, Pattern Recognition and Data Structure, May 1975, pp. 291-320, University of [illegible] Press, Ann Arbor, 1975.

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

Publications, 1970-1974

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

Brick, Edward W., A Data Structure [illegible], Proc. of Electrical Engineering, June 1971.

Crocker, Stephen D., Interprocess Communication on the ARPA Computer Network, S.B. and S.M. thesis, Dept. of Civil Engineering, June 1971.

Cutler, Scott B., Computer Graphics, Tech Engineering News, Vol. 52, No. 9, Pp. 14-21, 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, pp. 270-279, Irene Traylor (Ed.), Prentice Hall, Inc., Englewood Cliffs, N.J., 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.

MODELING, GRAPHICS, NETWORKS

Publications (cont'd.)

[illegible] Computer Graphics Through Pictorial Display Information for Higher Education, Michael A. Cowan, Edward T. McCarthy, and Stanley N. Reinhardt, Spartan Books, Inc., [illegible], 1970.

[illegible], Robert [illegible], Computer Graphics, Computer, Vol. 7, [illegible], Dec., Nov. [illegible], June 1971.

Vezza, A., and Engelbart, Douglas C., Procedure Documentation, Terminals, Computer Modelling and Data Information Processing, A. B. Murray (Ed.), Jun. 1970.

EDUCATION

Prof. R. R. Fenichel
Miss S. Weinbaum

Interpreters, Research Associates, Research Assistants and Guests

H. Kaplan
S. A. Ward

Undergraduate Students

R. H. Brown
D. M. Christ
S. P. Silberstein

Support Staff

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

Guests

J. Berger, Prof.
P. Eisenbert

[illegible]

[illegible]

The [illegible] Progress Report [illegible] at [illegible] describes the language, [illegible] system, which then was called [illegible] designated UNCL (Uncommonly Clean [illegible]

> "It is an interpretive language that somewhat resembles JOSS. It differs from JOSS and other JOSS like languages in several major respects. For example, the language has block structure, a context editor, and a function definition 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. 175-179.

Fenichel, Robert R., "List-Tracing in Systems with Multiple Cell-Types", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation (March 1971), pp. 241-247.

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).

PRECEDING PAGE BLANK NOT FILMED

R. G. Babcock
M. H. Brockman
[illegible]
[illegible]

Support [illegible]

[illegible] Yale

PRECEDING PAGE BLANK NOT FILMED

IX. IMPLICIT COMPUTATION

A. Introduction

This research concerns novel machines and algorithms for the solution of large-scale problems, e.g., [illegible] systems of equations. Two approaches are taken. In the first the problem is separated into two parts, one of which is computed accurately (i.e., digitally with a large number of bits) and an interrelated second part which is computed approximately (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 of linear systems of equations via iteration of the exact-inexact computing cycle.

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 at 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 Aanderaa on Turing Machines for the real numbers and one by Cartwright 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 processors to general-purpose machines, and as [illegible] to conventional digital computers. These applications are briefly discussed, along with the [illegible] results, in the following sections.

IMPLICIT COMPUTATION

B. Exact-Inexact Machines and Approach

[illegible] error $\varepsilon = x - Ax$ is computed exactly. It is then [illegible] and converted to an inexact value by truncation of the [illegible] least significant bits. This computation is performed in order to produce [illegible] the [illegible] error to a [illegible] of the [illegible] [illegible] converted to the value of [illegible] this [illegible] to the [illegible] the quantity $x + \Delta x$ [illegible] "corrected", after the correction vector has been computed by the exact subsystem, through multiplication of that vector by A^{-1}. Naturally, the truncated correction vector is applied through [illegible] sufficient number of cycles to effect truncations [illegible] to reduce the word length of the exact [illegible] to [illegible] progressively closer to that of the computer. [illegible] each program cycle [illegible] over a system whose [illegible] near full scale. Observe also that [illegible] multiplication and division operations are preferred for [illegible] accuracy.

Observe next that, using straightforward techniques, the exact subsystem computes the error in time proportional to N^2 for an $N \times N$ matrix A, at a word length of N 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^2, where N is the word length of the inexact computer. Thus, this is the case since N operations are needed and [illegible] multiplication is dominant, requiring time N^2. Once the matrix is inverted, the time expended per inexact cycle is [illegible] since an $N \times N$ matrix multiplies an N-vector. At 2 bits [illegible], the number of exact-inexact cycles needed is of order N since at each iteration, the exact error is reduced by roughly 2 bits (recall that the inexact error is [illegible] scaled up by N). Thus, the total time for the [illegible] process grows as

$$N \cdot N^2 + \frac{N}{2}(N^2 + N^2)$$

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

C. Pressure-Flow Machines

In this approach, the computing structures and [illegible] consideration involve the spatial interconnection of computing elements which correspond to the individual relations (or equations) that make up the overall problem. These computing elements, in turn [illegible] variables, which are called "pressures" and "flows" corresponding to the main variables [illegible] and [illegible] the [illegible] [illegible] [illegible] physical [illegible] We shall refer here to any [illegible] of flows.

variables related [illegible] the other hand, [illegible] satisfy the [illegible] each computing [illegible] as outputs [illegible] part of [illegible]

Stated, the flows go [illegible] pressures [illegible] ent by which the flows [illegible] relations [illegible] related to [illegible] through a pseudo-energy [illegible] of the relations [illegible] variables and conservation [illegible] energy-like [illegible] well as [illegible] properties of the pressures and flows are realized upon [illegible] position of the computing elements into larger [illegible] [illegible] — that is, the resultant systems have flows [illegible] and pressures for outputs, [illegible] related by the [illegible] of pseudo-energy constraints. The result of [illegible] is the ability to construct arbitrarily complex, partially [illegible] structures of pressure and flow variables that are [illegible] and that are capable of converging asynchronously to [illegible] solutions. The same sense that aggregates of passive [illegible] trical 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 (a 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 in terms of the 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, in the sense 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 [illegible] and (recursively) of composite relations [illegible] as follows:

a) External variables (i.e., free variables of the composite relation) appear as [illegible] pressures of the composite [illegible]. As [illegible] connected in the constituent machines, [illegible]

corresponding two variables are related by constituent relations. As a result, flows are generated by addition from corresponding inputs of all constituent machines which relate these variables.

b) Internal variables, i.e., variables which by the composite relation, which determine flows and pressures of the composite machine, are obtained, each such flow is uniquely determined by direct integration of the sum of all corresponding pressures supplied by constituent machines, i.e., the machines corresponding to constituent relations that relate that flow variable.

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

Under these composition rules, it can be shown that

a) the flows of every composite machine are those 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.

It follows 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 2 (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 as the flows of that composite machine are held constant. It has further been shown that for [illegible] pressure-flow machines which are of singular [illegible] the pseudo-energy obeys a law corresponding to [illegible]. These results make possible the application of pressure-flow machines to problems of [illegible]

d. Fundamental Work

The pressure-flow approach and the [illegible] concept machine are made up of [illegible] distributed systems. We have [illegible] mathematically [illegible] packing [illegible] any computing device, detects or stores the [illegible] tion of information. The principal result of our [illegible] date is that the time-decoupled [illegible] to compute any [illegible] function given [illegible] [illegible] This result is based only on the above-mentioned [illegible] and on the fact that the computing function depends non-trivially on all its arguments. Further research shows that, regardless of the way in which [illegible] [illegible] are "stacked" or [illegible], they cannot compute [illegible] [illegible] asymptotically [illegible] the above bound -- in fact, they [illegible] compute such a function no faster than [illegible] 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. Further this measure is possible to assess the efficiency of a given [illegible] process. The details of this development will appear in the Proceedings of the 1971 Switching and Automata Theory Conference in a paper by Cortouzos.

Another area of fundamental work is motivated by the logical capabilities and limitations of the analog substructure of an [illegible] 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. A. Faiski, M.I.T. Department of Electrical Engineering, to be completed shortly.

We have also investigated the logical capabilities of [illegible] of Turing Machines which can store and process real numbers. Results of this work are related to computation on the real numbers. They will appear in considerable 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 Digital Computation, S.M. Thesis, Dept. of Electrical Engineering, February 1971.

Cortouzos, M. [illegible], "Computer Graphics - Problems and Trends", Proceedings, Erlangen Symposium on Display Systems for Machine Design, Institute for Machine Tools and Production, Erlangen, Germany, May 8 1971.

Cortouzos, M. [illegible], "Iterative Systems and Computation: A New Experiment in Computing Networks", Digital Systems and Numerical Techniques, in the first Course of Theory and [illegible]

IMPLICIT COMPUTATION

Publications 1970-1971 (cont.)

1971 Symposium on Application of Computers to Engineering Education, Purdue University, Indiana, August 25-26, 1971.

Dickens, M. W., Computer Graphics [illegible] and Their Treatment, S.M. Thesis, Dept. of Electrical Engineering, June 1971.

Eng, M., Computer-Aided Analysis of Solid State Circuits, S.M. Thesis, Dept. of Electrical Engineering, January 1971.

Fynn, G. K., Nonuniform Function Processing for Computer Analysis of Networks, S.M. Thesis, Dept. of Electrical Engineering, June 1971.

Heinberg, W., Computer-Aided Education Instruction, S.M. Thesis, Dept. of Electrical Engineering, January 1971.

INTERACTIVE MANAGEMENT SYSTEMS

ORGANIZATIONAL INFORMATION SYSTEMS

Prof. W. M. Jones

Instructors, Research Associates, Research Assistants and Others

D. Asthana
G. T. Dixon
R. C. Goldstein
S. P. Mason
A. H. Monroe-Davies
J. C. Cecil
L. H. Rogers

Undergraduate Students

R. L. Brooks
R. M. Elkin
R. S. Goldhor
F. H. Guldberg
C. A. Batvany
D. N. Brackbrook
W. Y. So
C. Pierce
J. L. Rosenberg
H. J. Siegel

USM Staff

G. A. Friel
M. Berol
S. E. Niles
A. J. Stenad
D. M. Wells

Support Staff

E. T. Moore
R. Queens

Guest

Prof. J. T. Elkind

* * * * * * *

SIMPLE PROJECT

Prof. W. M. Jones

Instructors, Research Associates, Research Assistants and Others

R. Bryant
A. Gonzales
S. Murthy

Undergraduate Students

[illegible]

USM Staff

[illegible]

PRECEDING PAGE BLANK NOT FILMED

X. INTERACTIVE MANAGEMENT SYSTEMS

ORGANIZATIONAL INFORMATION SYSTEMS

A. Introduction

The Organizational Information Systems Group has as its goal 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 situations, 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 processes of Project MAC and of several 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 MAC-IMS 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 additional information systems for Project MAC on CTSS. Some progress was made in developing interactive systems for personnel management, budgeting, inventory control and purchasing, but little lasting value has resulted 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 for modeling organizations.

[illegible]

INTERACTIVE MANAGEMENT SYSTEMS

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 operational 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/I 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. M. Stream and E. S. Niles, uses the data-manipulation system described above. In its initial version, the system will provide these services: data storage, retrieval, manipulation, and report generation support for the personnel-management, budgeting, inventory-control and facilities-management functions of the [illegible]

offices. 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. D. Pekham, 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 present 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

Richard'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 in 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.

B. C. Goldstein has begun a doctoral thesis, "The Political Economics 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

R. Kreindorfer, M. Treyor, M. D. Wiegel, and Prof. [illegible] 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", Vol. 3, No. 5-6, 3rd Views, January 1971; also Project MAC Technical Memorandum 25, April 1971, AD 721-592.

Goldstein, Robert C., "The Assessment Value of Computers for Intellectual Activities", Project MAC Technical Memorandum 24, April 1971, AD 721-619.

Goldstein, Robert C., and Strnad, Alois J., "The MacAIMS Data Management System", presented at the ACM SIGFIDET 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 728816.

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 722-877; [illegible] accepted for presentation at IFIPS, August 1971.

SIMPL PROJECT*

A crude version of SIMPL was initially available by the beginning of August 1970, but it had not been thoroughly tested and many of the advanced features of SIMPL were not available. In August, the system was used by 30+ graduate students in a special Sloan School Summer Session Simulation course during which period we discovered many of the inadequacies 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 (15.472) 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. Queueing statistics and a limited 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 prolonged the already-long translation and compilation times.

Thus, the system was not fulfilling 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 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 coding beginning in February 1971, and is now in [illegible] testing and debugging. [illegible] being completed [illegible] lation, compilation, execution, [illegible] includes a [illegible] tracing capability, allows external activities [illegible] or simulation-oriented statistics, and supports the interactive SIMPL Monitor. The old Version 1 was [illegible] earlier version 1. The SIMPL Monitor itself is [illegible] ing; it is a very flexible run-time system which allows almost complete freedom to inspect and modify [illegible] to continue or restart the simulation.

The SIMPL system has also been concurrently 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 manuals, 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 graphics facility to SIMPL, using something like the IMLAC IDS-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 slow 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 [illegible]

MACSYMA

Prof. W. A. Martin
Prof. Ju Kung

Instructors, Research Associates, Research Assistants and Others

R. J. Fateman
T. S. Wang

Undergraduate Students

M. R. Genesereth
E. Kehl
B. C. Rosen
J. F. Saunders
E. Tsiang
L. I. Widman
R. F. Zippel

USR Staff

M. J. Ablowitz
L. O. Cairo
J. P. Golden
L. P. Rothschild
R. C. Senfrappel

Support Staff

K. Young

X. MACSYMA

During the past year the Martin group has continued development of the MACSYMA system for intelligent algebraic manipulation. The principal modules of MACSYMA are shown in Fig. 1. Those indicated by circles are complete.

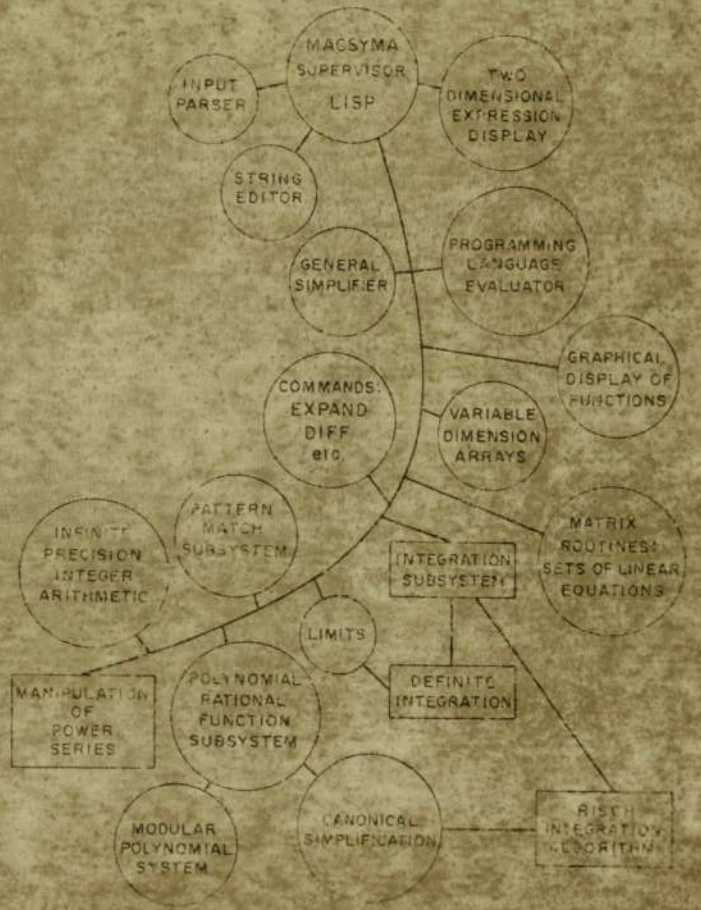


FIG. 1 THE MACSYMA SYSTEM.

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, March 1971. MACSYMA now reached a point where it is both a useful tool for the solution of real problems and a convenient base for research in algorithmic analysis and implementation of advanced 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. P. Noble's S. M. thesis (M.I.T. Physics) were done in MACSYMA.

Within our own group, D. 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 or functions, defined by differential equations, are also being developed.

Publications 1 70-1971

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.

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, "Symbolic Integration", 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 Integrals", in Second Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, Los Angeles, California, March 23-25, 1971, pp. 458-464.

PRECEDING PAGE BLANK NOT FILMED

Los Angeles Chapter

Prof. [illegible], Director

Instructors, Professors, Research Associates and Others

V. Altman
[illegible] D. McCarthy
[illegible] Chapin
[illegible] Goodman
W. Heise
[illegible] Johnson
W. [illegible] King
D. [illegible]
[illegible] Mazwick
[illegible] Nichols
[illegible] Olsen
H. A. Young
L. E. Travis

Undergraduate Students

F. G. Drake
[illegible] Lewis
C. A. Reiser
W. Vaughan
J. C. Bland
J. Gurney
J. [illegible] Sandy
A. M. Sollish

Support Staff

[illegible] Fulcher
J. E. Vanoli

Guest

M. Adler

PRECEDING PAGE BLANK NOT FILMED

[illegible] Programming Languages

A. Introduction

During 1970-1971, [illegible] the Programming Language [illegible] focused on analysis of [illegible] translators (compilers) and the environment in which they operate (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 Post'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 (assertions) and the logical rules of substitution for variables and detachment (modus ponens). A canonic system defines a set of [illegible] productions, each of which is a pair 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 rules

D (P,V) is a set of sentence productions, 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 (to provide serious progress in the direction of [illegible] and [illegible] formal properties, and study how they can be [illegible] by [illegible] scientists and engineers).

C. Power of Canonic Systems

We have proven a [illegible] theorem relating canonic systems to [illegible] languages.

3 OF 4
7225224 UNCLAS

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 which generates the same language and that belongs to the class. Further, that class of canonic systems is [illegible]

Turning to the systems -- for example, canonic systems and syntax grammars -- have inherent decidability properties in general there is no algorithm for telling, after a finite amount of time, whether or not a given string is in the language of these grammars. Studies of these properties 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.

5. Canonic Systems and Recursive Sets

We have proved 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 1 define all and only languages of Type 1; 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 $\subsetneq$ PSC. The proof is by diagonalization (after a suitable "G\"odelization" of canonic systems).

6. Generalized Translator

An efficient algorithm, which is capable of recognizing the sentences produced by a canonic system has been developed. It can easily be seen that an earlier algorithm presented by Chomsky and Schützenberger 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 eventually at a lower-level predicate (e.g., integer or digit).

This algorithm removes the inadequacies of earlier Post Canonic "BNF" specifications of the syntax of programming languages, and it is impossible to describe many of the constraints that

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

The stream of statements given as input to a PL/I compiler contains statements to be compiled and statements addressed to the preprocessor to be executed at compile time. Confining our attention to a small but [illegible] program written in a language which includes control transfers of control, it is decidable whether there are no 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/I 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

[illegible] the interaction between programming languages and the

operating system in which it [illegible] distinct, we find [illegible] group. The most important [illegible] that the programming language [illegible] ...

We have developed an approach [illegible] system that [illegible] ... really supplement [illegible] ... not be used both for [illegible] ... of the language.

These ideas have been [illegible] ... conditioning [illegible] ... statement of [illegible] ... end, we have developed an advanced [illegible] ... system for the IBM 1130.

This included [illegible] ... mechanism, [illegible] ... Advanced Linker, and [illegible] ...

J. Community Activities

Members of the group were involved in two major community activities. We feel that M.I.T.'s greatness and many generations of 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 MIT students.

The other group to which we have addressed [illegible] underprivileged people in the Boston [illegible] We have addressed ourselves to this group through the Learning School, which is a school under the auspices of M.I.T. [illegible] ... We have helped to restructure the school [illegible] and teach these people and have exposed them to the process of learning and the rewards and satisfaction [illegible] ... We have centered the program around computers, using them as a tool for accomplishing our objectives.

K. Teaching

Many of our [illegible] ... [illegible] ... the basic [illegible] ... language ...

PRECEDING PAGE BLANK NOT FILMED

Course [illegible] - Programming Languages in Formal Systems
3 Graduate Credit Courses

Course 2 - Operating Systems - Independent Activities
Period Seminar

Course 3 - Digital Computer Programming Systems - Undergraduate

Publications 1970-1971

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

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

Farley, 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.

Klug, 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 Senior 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.

Ramchandan, Chan. P., "Debugging Screens to Run Interpretively in Virtual Memory", S. M. Thesis, Department of Electrical Engineering, January 1970.

Travis, Leon F., "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.

[illegible]

Publications

[illegible]

July 1970 - December 1970

Prof. A. Ninke
Prof. S. Parker

Academic Staff

Prof. A. J. Fischer
Prof. R. P. Gabriel
Prof. S. Parker
Prof. N. S. Peterson
Prof. P. Wilner

Instructors, Research Associates, Professional Assistants and Others

H. Abelson
E. Chafnina
R. J. Dougherty
M. Dowson
B. Freuder
I. Goldstein
B. K. P. Horn
L. Krakauer
R. LeCompte
L. Lerman
S. Lerner
R. A. Roberts
M. Shoriche
D. Spencer
G. J. Sussman
S. Waltz
J. R. Winograd

Undergraduate Students

D. T. Dalton
W. Freeman
J. Freiberg
P. Gagner
J. Gaschnig
S. Glazer
R. Goodwin
J. C. Holzer
P. Jonken
R. F. Mohl
J. Rubin
J. C. Shockley
M. N. Slimarozuk
E. M. Traeger
L. D. Travis
S. S. Weinstein
M. Weitzman
L. Zuckerman

DSR Staff

M. Bueler
R. H. Freyberg
D. W. Cooper, Jr.
P. Greenblatt
A. K. Griffith
W. Henneman
T. F. Knight
W. Neely
P. Wolfsker
[illegible] Roe
L. P. Rothschild
P. R. Samson
F. Schroeppel
J. Shah
S. W. Sholar
C. Solomon
G. L. Wallace
J. L. White
S. A. Williams
[illegible] Wright

ARTIFICIAL INTELLIGENCE

Support Staff

T. E. Callahan
T. Carlton
P. DeGriselle
[illegible] Grenckhahn
[illegible] Hostlake
J. L. Knight
M. Margolo[illegible]
[illegible]
[illegible]

P. J. Lehei
C. H. H. Mitchell
G. Rod[illegible]
L. A. Rose
D. Silber
N. L. Stone
O. T. Sullivan
J. R. Weiss

Guests

H. Abril
Prof. W. W. Bledsoe
P. Bayer

J. Cohen
J. Jarmasz

1. ARTIFICIAL INTELLIGENCE

The A. I. Laboratory is concerned with understanding the principles of intelligence. Its goal is to develop a [illegible] program that [illegible] the [illegible] of [illegible] Natural Intelligence, and theories of computation [illegible] to be main current [illegible] of [illegible]

ARTIFICIAL INTELLIGENCE

Robotics; Vision, mechanical manipulation, knowledge formation. Models for learning, induction, analogy, [illegible] organizing bodies of knowledge. Development of hardware program control structures.

NATURAL INTELLIGENCE

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

THEORY

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

[illegible] Our [illegible] Artificial Intelligence [illegible] The natural language project is interwoven with the robot [illegible] and reasoning study, in turn essential to the computer [illegible] ing 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 is the [illegible] puzzle, game, and theorem-proving environments which were traditional in the early years of Artificial Intelligence research. Our evolutionary 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 [illegible] 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 crystallinely:

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 such [illegible] [illegible] PROCEDURAL CONTROL structures [illegible] which control [illegible] [illegible]

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 computations. By using "know" we are not saying that an intelligent machine, or person, must have such knowledge available at the level of overt statements or conscious access, 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 the 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

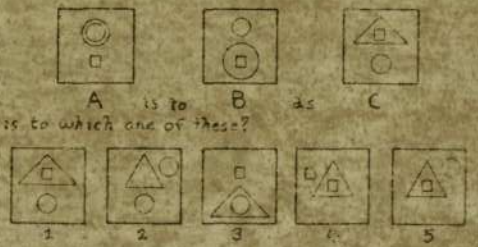
When we enter a room, we feel we see the entire scene. Actually, the peripheral part of it is out of focus, and poorly imaged; our peripheral vision is weak in detail and color; one sees nothing in his blind spot; and there are many things in the scene we have not understood. It takes a long time to find all the hidden animals in a child's puzzle picture, yet one sees 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 analyze one's own mental processes efficiently 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; [illegible] Vision [illegible] able to result to make meaning [illegible]

[illegible]

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 we develop it will become clear that [illegible] trying to distinguish "sensation" or "perception" as separate and different from other aspects of thinking.

When we give an "educated person this kind of problem from an IQ test, he usually chooses the answer "3" over "5":



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 shape 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 answer, 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 [illegible]

[illegible]

identical in essence although they differ in details.

It [illegible] in [illegible] over the possibility of many other such statements [illegible] choosing different [illegible]. 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 [illegible]," etc.

And [illegible] that [illegible] statement were [illegible] also to C and [illegible], 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 [illegible] 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 [illegible] was the [illegible] behind a computer program written in 1963 by T. G. Evans. Its performance on "standard" [illegible] analogy tests was comparable to that of fifteen-year-old [illegible]. 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 [illegible] with experience. Certainly, we cannot propose it as a complete model of "general intelligence". [illegible], analogical thinking is a vital component of thinking, [illegible] theory [illegible] 1964), [illegible] important step.

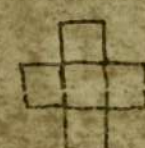
In developing [illegible] schematic outline [illegible] complete computer program, [illegible] to fill in [illegible] detail: one must decide on ways to describe [illegible] to change descriptions, and [illegible] to describe [illegible] also have [illegible] for deciding when one [illegible] "fits much better" than [illegible] available variety of possible descriptions [illegible] [illegible]; how can we [illegible] descriptions should be [illegible] problem. Try [illegible] tion between A and B that might be [illegible] resources of a child and you will see that [illegible] beyond simple combinations of a few phrases like "inside", "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 theme: 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.[illegible] for pictures and in 5.[illegible] for verbal descriptions of [illegible] situations. In 5.6 there are some descriptions which are themselves 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.

[illegible] Children's Use of Descriptions

The theory of analogy we have just proposed might seem to be 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 [illegible] information.



We asked a [illegible] to draw a cube. This is what he drew. "Very good," we [illegible] "How many [illegible]" "[illegible], of course," [illegible].











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 pose is no better, optically, than the old, but the child is delighted to learn a symbolism to depict protrusion.

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 repetitious and tedious "schedules" of reward, deprivation and punishment. There is only a minute amount of attention to the kind of "one-trial" experience (such as when you tell a child how to [illegible], or in which he asks you what [illegible] if you tell a child just once, that the elephant in [illegible] and don't ask again a year later, he may [illegible] that you do not.

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

It is natural to ask whether human brains [illegible] descriptions or, instead, manage somehow to [illegible] with something closer to the original optical [illegible] tion in view of today's limited understanding [illegible] works. Nevertheless, the formalistic [illegible] children's drawings point clearly toward [illegible] The phenomena in the drawings suggest that they [illegible] rather small variety of elementary object-symbols, [illegible] in accord with a few kinds of relations involving [illegible] perhaps taken only one or two at a time. [illegible] not seen so clearly in the pictures of [illegible] but even so we think the difference is only a matter of degree. While it is possible to train oneself to [illegible] with quantitative accuracy, some aspects of the "true" [illegible] difficulty or learning this is itself [illegible] symbolic [illegible] is the more normal manner of [illegible]

sophisticated adults often show a preference for unrealistic but tidy, "isometric" drawings over more "realistic" perspective drawings.



(i)

Even though a cube is never seen exactly as in (i), in any case, all this suggests that "graphic" visual representations are more operative later on at early ages than 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 mature [illegible] maturation.

2. Appearance and Illusion

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



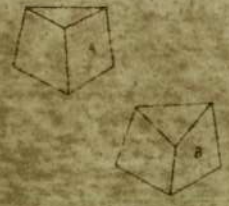
But on the right, the diagonal stripes affect its appearance so that (to most people) the sides appear to [illegible] and the ends seem perfectly parallel. Such phenomena have been studied with great intensity by psychologists. In the [illegible] figures, the central squares actually have the same proportions, but everyone sees the one at the left as darker.



Much [illegible] 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.






A
B







points on axis	sides [illegible] axis
one point on each side	two [illegible] on [illegible] side
made of two triangles	made of two rectangles
unstable on ground	stable -- flat bottom
hurts when squeezed	[illegible]

These two descriptions could hardly be more different! [illegible] 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 symmetric axes. [illegible] is then almost [illegible] whatever [illegible] [illegible] set up [illegible] spatial frame of reference for each group.



Since one has to choose an axis for each square and "other things being equal" [illegible] locally [illegible] [illegible] choice one tends to use the axis inherited from the direction of its "row". The fact that [illegible] if you [illegible] choose to see any of the symbols as either diamond or square only confirms [illegible] -- the choice is by default only, and [illegible] would be expected to carry little force.

[illegible] does [illegible] [illegible] other [illegible] [illegible] elements [illegible] -- as well as on other things in the picture. [illegible] a face, a circle can be an eye, a mouth, [illegible] the whole head. There should be no difficulty in [illegible] -- or to the [illegible] programs that [illegible] and performance. [illegible] [illegible] (rather than on computational or symbolic mechanisms) [illegible] inherently unable to [illegible] [illegible] and [illegible]

2.2 Sensation, Perception and Cognition

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

WORLD → SENSATION → PERCEPTION → RECOGNITION → COGNITION → ...

[illegible] to explain exactly what these stages [illegible] are, everyone comes to believe that they exist. [illegible] thinking rejects the idea that there [illegible] separate activities like "perception" that precede and are basically independent of "higher" intellectual activities. What one "sees" depends [illegible] one's current motives, intentions, memories, and acquired processes. [illegible] to say [illegible] that the old [illegible] 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 [illegible] and this might suggest a basis for the popular [illegible] 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 [illegible] 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. [illegible] tional view of a mechanism as a HIERARCHY of parts, sub-assemblies and sub-sub-assemblies (e.g., the main bearing [illegible] of the pitch [illegible] pocket of the [illegible] give way to a HETERARCHY of computational [illegible].

It is unfortunate that technical [illegible] guidelines, for such heterarchies [illegible] The reader [illegible]

[illegible]

A [illegible] current [illegible] in the [illegible] [illegible] important [illegible]

technical means to give it the appropriate degree of precision. Consequently, the dividing line becomes prematurely entrenched in the mythology of artificial intelligence. The concept of "perfect" is used in attempts to differentiate between the computer and the people it models, and these efforts often lead to misunderstanding of the whole enterprise. [illegible] a theory whose [illegible] much the kind of [illegible] might be considered [illegible] in [illegible] "nothing but" [illegible].



But the arch has properties -- as a whole -- that are not manifested directly in the properties of its parts in any simple way. Some of these 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 triple of line segments 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:

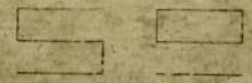


SQUARES

[illegible] about [illegible] not really [illegible] about [illegible] about how one might "parse" [illegible] easily detected features [illegible]

geometric form. But there is no [illegible] kinds of problems that arise when one looks at [illegible] visual scenes.

The next two figures are [illegible] in the following precise sense. Imagine innumerable experiments in 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 evoke precisely identical collections of such records -- provided that we keep the records of the two pictures separate. In this sense both pictures have the same "parts". They also 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 [illegible] about a picture from such "spatial" local evidence. But this is not true. For example, it can completely determine the property of being "one-entirely-of-separate-solid-region" by merely knowing that all [illegible] one or another of those micro-scenes



[illegible] a simple [illegible] a convex [illegible]

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

EXTREMELY ATOMIC ALGORITHM: One of the input variables is connected to the output, the others are ignored.

VETO ALGORITHM: If every input says "yes", the output is "yes". If any input says "no", the output is "no".

MAJORITY ALGORITHM: If more than half the inputs say "yes", output is "yes".

LINEAR SUM ALGORITHM: To each input is associated 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 if every pair has the same number of "yes" 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. Each such a class is defined by choosing a meaning for the phrase "very simple function". For example, one might specify that each 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 are of this kind. The next example has no "order restriction", but the function

are very simple in another sense: they are the "linear one" or linear-threshold functions.

GAMBA PERCEPTRONS: A number of linear threshold systems have their outputs connected to an output linear threshold system. This can have the form of a threshold function of many linear 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 is very much an opinion, not a fact, although we could prove some relations between the orders of a picture, that could not be detected by any relations of any significant size. 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 are not? Rather than muddle through a philosophical discussion of which classes "really" do more than add the parts, we should try to classify the kinds of mechanisms needed to realize each interesting "hardware" framework chosen for good mathematical reasons. Then for each such framework, we might try to see which of the simple reinforcement mechanisms for learning, which admit efficient cooperative learning (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 of systems. One should not expect to handle complex systems until one thoroughly understands the phenomena that emerge from their simpler subsystems. This is why we focus our 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 mathematical-psychological terms like "linear", "independent", or "Markov process", etc., seem not (we hope) profound, that is because memory is one of those things we understand as part of a well-developed technical vocabulary for describing the structure of more complicated automata. But until recently there was a serious shortage of ways to describe more specific 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 to recite a family of concepts to support effective theories of intelligence, learning, and development. Neither the finite-state and stimulus-response cartoons of the behaviorists, the hydraulic and economic analogies of early Freud, the holistic insights of the Gestaltists applied to the neurological ingredients to develop such an intricate system needs a substrate of [illegible] and [illegible] but [illegible] perceptrons, [illegible] and [illegible] blocks [illegible] will define [illegible] the [illegible] only a fraction [illegible]

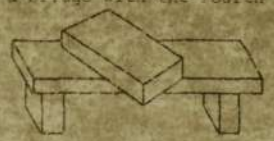
gnishable representations in traditional psychology:

symbol	[illegible]
procedure	pushdown list
time-sharing	interrupt
calling sequence	[illegible]
"machine" language	[illegible]
memory [illegible]	[illegible]
dispatch table	hardware-software trade-off
[illegible]	serial-parallel trade-off
[illegible]	time-memory trade-off
branching	conditional branch
formal language	[illegible]
compiler	interpreter
indirect address	[illegible]
macro language	list structure
property list	block structure
data type	look-ahead
hash coding	look-behind (cache)
[illegible]	diagnostic program
format matching	executive program
syntax-directed	operating system

[illegible] only a few ideas from the environment of general "systems programming" and debugging we have mentioned [illegible] much larger set of concepts specifically relevant to programming languages, artificial intelligence research, computer hardware design, and other advanced and specialized areas. All these serve today as tools, crutches and intricate craft in 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.

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 part "lying across it" indicates knowing that the block is actually resting on the bridge. Indeed, in the pronoun "it" we refer to the bridge, rather than to the top block of the bridge, where is implied a further level of functional analysis.

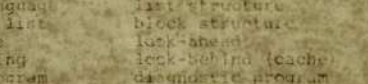


FOUR BLOCK BRIDGE

In our exhibit program's reports we described the [illegible] program (Guzman, 1968) which was able to assemble [illegible] correctly segments of the outlined regions of this picture into four objects, using a variety of criteria to "recognize" bridges.

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 "bridges") to yield just the kind of functional description we are discussing. We shall see program is even able to learn to recognize such a physical using experience with examples and counterexamples, 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 1948) it was felt that the hardest problems in apprehending a 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 object recognition, this kind of problem is usually handled by a two-step process in which the image is first "normalized" to a standard position and then "matched" -- by a correlation or filtering process -- to one of a set of standard templates. In practical engineering applications, the [illegible] is often fallible because it could not discriminate parts of images that touch together, and "matching" often fails 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 external 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 to [illegible] surveying 1959. But we will not stop to review the field of character recognition, for its technology is [illegible] problems that concern us more like how it is [illegible] that [illegible] is recognized correctly [illegible] (either by [illegible] or by [illegible] parts of

their end surfaces, simply do not occur at all in the two-dimensional case. Some more interesting two-dimensional forms require description when generalizing to three dimensions. For example, "a rectangle" is not simply a parallelogram with four right angles:

"Two pairs of congruent lines
adjoin, connected so that
third line."

3.1 Program for Finding Bodies in Scenes

Let us review briefly Guzman's SEE program, which is a conjunction of "local" programs that operate directly on the optical data. Their job is to find connected regions of homogeneous color and texture so that the scene can be described in a simple way in the program's data-structure. Next, the regions are classified into "bodies". The most important kinds are these:



ARROW FORK T L

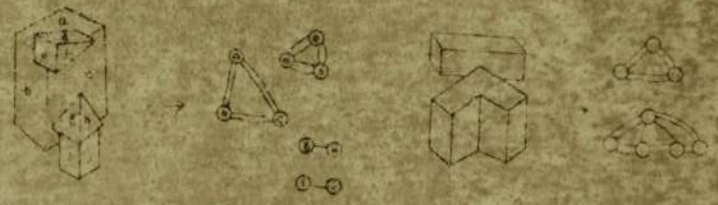
The main goal of the program is to divide the scene into "objects" and its basic method is to group "regions that probably belong to the same object. Each type of vertex is considered to provide some evidence about which regions may be used to create "links" between regions.

For example, the ARROW type of vertex usually is caused by the intersection of an object, where two of its plane surfaces form an edge. So it inserts a "link" between the two regions that adjoin both of the smaller surfaces.





Similarly, the FORK type of vertex, which usually has three planes of the object, causes links between three regions.



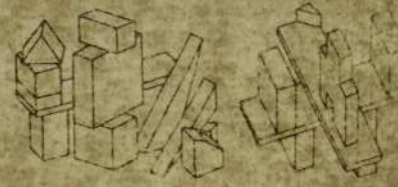


Many scenes are handled correctly by just these simple rules, but many are not. For example, the basic assumption about the FORK junction's three regions is not true of concave corners, and the "matching TEE" assumption may be falsified by coincidence, so that "false links" may be produced in such cases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

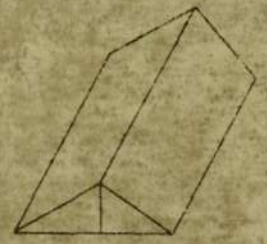


Guzman introduced several methods for correcting such errors. One method is to look for consistent groups of links, and another is to look for [illegible] qualitative [illegible] Typically, these methods create [illegible] errors but sometimes [illegible] are [illegible].



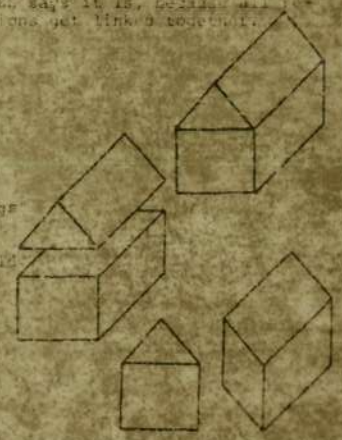


But SEE has faults, among which are:

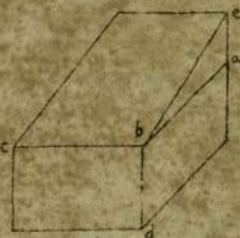


ORDINARY VIEWPOINTS: Curved simple figures are not handled correctly. Some figures (any scene with a cone, as could conservatively be true of many objects) cannot be analyzed. The goal is to find an analysis that makes sense in everyday situations. Normally one would not suppose that this is a single body, but SEE says it is, because all lines 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 pair 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.)



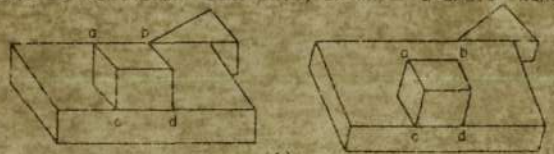
Following Guzman's work, Martin Rattner has described a procedure, called SEEMORE, that can handle some of these problems. (Rattner, 1976). While it uses linking heuristics much as did Guzman, SEEMORE puts more emphasis on local evidence than an edge might be part of two bodies. The SEE splitting heuristics operate initially at certain kinds of vertices, roughly trivertices and vertices with three or four 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 a pick-up list so that they can later be used to separate out multiple bodies.



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.

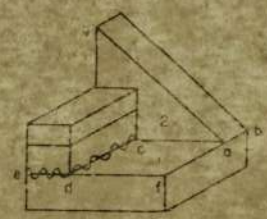
The vertex 'a' strongly suggests a split along a-b, while neither 'c', 'd', nor 'e' have much in their favor. Thus SEEMORE starts a split at 'a' and continues at 'b' toward 'c'. Generally, splits originate at TEE's, propagate through L's and matching FEE's, and avoid the sharpest turns through the multiple-edge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

Degenerate situations like this, in which a small change in



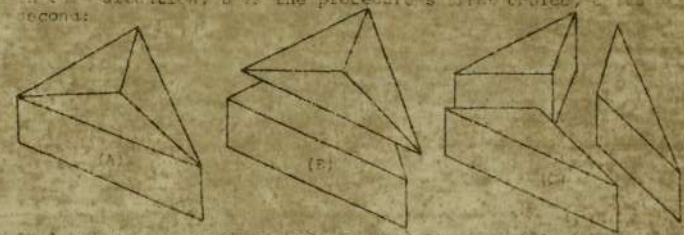
viewing angle produces a different topology, are likely to lead to "incorrect" analyses. Rattner uses a rather conservative linking phase, in which links are placed more cautiously than in SEE, but using similar "inhibiting" rules. Regions that are doubly linked to one another by these are considered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

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

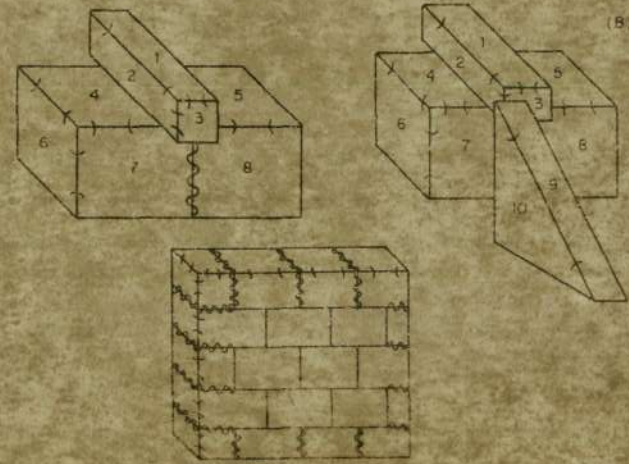


An initial split is made along c-d, extended to b-e. Then between the possible splits a-a-1 and e-e-1, the latter is preferred because it completes the trihedral vertex at

In this situation, B is the procedure's first choice, C is 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.



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, from 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 problem, or trying to fix something). Here the micro-world consists of the arrangements of squares on the chessboard that arise when one moves the pieces. Solving the chess problem consists largely of coming to know the relations between the pieces, and how the moves affect things -- using naturally such words like "after" in this context. As the problem goes on, one experiences events in which one suddenly sees certain relations. A group of pieces is seen as three pieces playing different roles; it 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 these 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 such, and the problem of learning will vanish. In the last, decisive stages of experience with learning systems that succeed dramatically, although we did not build such a system, this was the case. We believed that learning was not the problem of determining what was best, but rather to develop [illegible] the intelligent [illegible] describing, analyzing processes. For the purposes of such languages as [illegible] such a mechanism, in favor of studying systems that could develop [illegible] fragments of reasoning, and [illegible] of uninterpreted [illegible] as a way of analyzing the interactions of [illegible] we [illegible] and experiment.

In the rest of this chapter, we will discuss [illegible] 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 critically different from the static structural situations we already understand rather well. Our position is intermediate between those, 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.

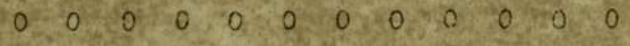
Consequently, we are in a poor position to estimate how complex must be the initial endowment of intelligent learners -- one that could develop as rapidly as human minds rather than requiring evolutionary epochs. We certainly cannot assert from what we know that the "innate structure" required must be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

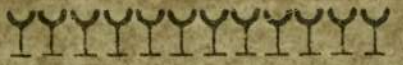
4.1 An example of learning: Piaget's conservation experiments

A classical experiment of Jean Piaget shows remarkably regular patterns of responses by children in the age range of 4-7 years. It questions [illegible] kind 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!"

Further 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 as it was" and have changed

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



In this experiment [illegible] confirm [illegible] the rods are equal at first, but when placed as shown at the right, the "upper" one is usually said to be longer.



How can we explain the difference between the [illegible] 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 the conservationist one? To many psychologists only the second is 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 perceptions," 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, that "appearance" theory is too incomplete: the notion of appearance is not structured enough. Secondly, we know that such 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 an, 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, chooses 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 readily tell which is "wider" or "fatter" when it is not a question of which is "more". Indeed, if asked the younger child is free to describe in whatever terms before asking which is more, he may say something like

(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 child has many things on her mind. His syntactic structure will not be so elaborate, but from the context, he is unlikely to produce that marvelous circumlocution in any one description. If we ask him "What are they?" he might say any of "Glass dishes", "Clear bowls", "Round", etc., depending on his various interests at the moment. As a purpose for the question of the experimenter, if we ask him for a description after he has drunk all the water which has been, 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 cokes 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 much? 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 I asked?
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 dominant mutually exclusive classes and the worst thing is to count any thing in two classes. So he proceeds to describe the situation first in terms of [illegible] purposes, and in this frame puts the [illegible]

It is [illegible] that the pre-occupation with "more" leads to the [illegible] question. We [illegible] might [illegible] to the [illegible] naturally, so [illegible] with the child than the [illegible] "trick" questions, anyway.

Returning to [illegible] "around" by which [illegible] "learning" process could cause a child to acquire this [illegible] idea? Our mini-theory begins not by trying to explain the particular fact (why the child says this about water poured [illegible] toys) but to look for a general trait for comparing quantities that combines simplicity with widespread utility. What is bigger: the child or his cousin? Should both get the same [illegible] do you divide a bottle of coke between two glasses? By the level -- and generally this is right because the glasses are identical. Finally, the child can afford to be wrong some of the time; this rule seems to work well enough, just as [illegible] it would be hard to [illegible] a better one without losing a valuable step.



[illegible] 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 tell-tale quality upon which to base its decision -- as is easily found in (B). If there is none -- as in (A) -- then a subroutine has to make a "quantitative" comparison. But even this seems to [illegible] opposite than our intuitive notion of how we compare "taller-higher" with "wider-broader", the former will surely [illegible]



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 quantities in two dimensions, this would be almost impossible for him.

We must now ask, then, what was the child doing, and how can we explain the transition from one to the next stage? The simplest theory would assert that the child specifically learns each concept (and, earlier, each perception technique) as a fixed piece of knowledge. However, such a theory is incomplete because it postulates some agency (unspecified) as responsible for the specific learning. A more satisfactory kind of theory would treat the child himself as the part of the "teaching agent" in our work theory, and ring his own attractive (but inadequate) descriptions for his problems.

Consider again the original conservation-of-number experiment. Suppose that we begin to talk the child how to behave. An authoritarian approach would shout at him, no, no, they are equal, but most teachers would prefer the gentler approach of explaining what he is doing wrong. One could say: "Jim, you are right, the dots take up more space than the dots, so you could say that A HAS MORE SPACE than B, but NUMERICALLY there are still the same dots in each row."

We have readers who are objecting that no child of five will understand what we are saying. Indeed, one can go a step further and say that it is the entire lesson begs the entire question. The pre-conservation child seems to lack a sharp distinction between "spatial" and "numerical". That's his problem. If he knew how to use the distinction well enough, he would not need to be taught him about conservation. Our child has already a wealth 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 disappear away. As it should! For "conservation" is not a single thing, and its 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 procedure:

(1) CHOOSE instead of rules. Choices are
SPATIAL-TOTAL SIZES
NUMERICAL-BUILD

(2) Try EXTENT. Select a kind. Choices are
EXTENT implies more
SPACEMESS implies less

(3) Try EXTENT. The spatial extent.

(4) Test for coherence with other DATA
Failed: Try SPACEMESS. The
spanned.

This does not
An inconsistency. Reject or explain.
Reject rejected

(3') Try NUMERICAL.
Try COUNTING
Don't know how to count.

Reject KIND

(2') Reject choice of quantitative rules.
Try the next choice, HISTORICAL

When HISTORICAL is tried, one might first choose
IDENTICAL. Some eggs were moved, but none added
or taken away.
Test for coherence with other HISTORICAL rules
Try EQUIVALENCY. The operation SPREAD-
OUT is reversible. This does not 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 will. This gives the right answer, because we 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 process for selecting descriptions; in what kind of environment would this operate? One kind of model would assume that the mature child's description is at first more elaborate, including both spatial and historical elements.

"The amounts of water in the first and second jars were equal. Then water was added to the second jar, so now in the third jar. Two waters in the third jar is greater and thinner than that in the first jar."

Later [illegible], we might [illegible] all its 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 tactics for such revisions.

Choosing among the many alternative methods for answering the question "Are they similar?" is to apply something more than the description itself. For example, when he chooses a "distant" perception 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 commonsense 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 not all right, then any decision 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 figures.

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 won't attempt to trade height with color, for example. What features of pictures 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 available geometric features must exist and be debugged well enough that it can be relied upon. The child needs to have and trust the hierarchy of 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! Moreover, 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 worth competing theories, not before one has any! It takes several years to work out any of this, and a theory that explains away too many problems is just to be distrusted.

We do not, we repeat, want to explain the difference between situations either on completely separate bases or by the use of one principle. We want to see it as the outcome of many processes in the child's procedures for dealing with the various descriptions that he comes into possession of.

In the result-oriented "theories of learning" there are questions like 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 more like:

"How can such-and-such a procedure be added to the descriptive or deductive system?"

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 7+4=11 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 lies in the discovery of descriptions that emphasize the "essential" aspects of things or events, and omit or suppress 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 and 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 magnitude of the problem of not getting lost in searching through gigabytes 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 itself, discovering things under the vague guidance of an attentive instructor, trying out a mediocre textbook, and having it explained "in principle" and completely by a superb expositor.

It is tempting to try to disentangle this messy web of [illegible] 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

1. Learning Without Description -- "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. With... the classes of concepts that these machines can represent, that is, describe as [illegible] "sums" of already programmed "predicates", the learning abilities are effective and interesting. However, for descriptively [illegible] these quasi-linear [illegible] 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 or to recognize circles. But the same machine would probably not be able to learn the class of "circles or squares"; it simply could not describe (hence learn) so complex a concept as "a circle inside a square".

These limitations are very confining. It is true that these methods can be useful in "decision-making" and "weighted decisions" in which many things are understood so poorly that "weighted decision" is better than nothing! But we think it might be useful to look at this in retrospective by a slogan: it is 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 segments 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 separations" 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 specifications, [illegible]

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 which although some of it is essential skill in playing chess, seems not to belong to any special principles of strategy or planning.

People working on [illegible] "adaptive learning" schemes seldom get to the programming of [illegible] because they are aware of this kind of problem. Unfortunately, most approaches to it take the form of attempts to generalize the sufficient optimizing schemes directly available for structures of the same kind, such as n-layer perceptrons. In doing so, one immediately runs into mathematical problems: no one has found suitable 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 using 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 response", 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 on 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-trial learning processes could yield an external appearance of slow improvement. In this kind of theory, a response is [illegible] connected with many different combinations of stimulus features or elements at a couple of [illegible]

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 how it gets to many trials to learn an isolated [illegible] sequence of acts, and more interested in how one can learn what a word means from hearing it only a single meaningful sentence.

What is the basis for the optimistic belief that a person can "memorize" something by rote? If a person takes the practice piece of music to play over and over, and we want him to learn to read music, 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 present so much as proposing and testing new ones.

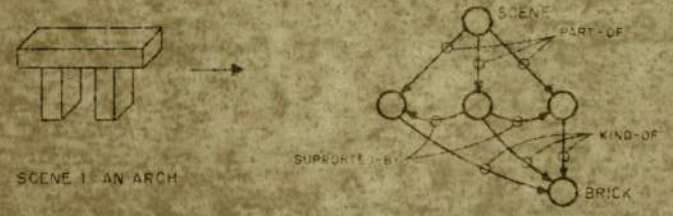
The external appearance of slow improvement, in the SYMBOLIC view, is an illusion due to our lack of discernment. Even practicing scales, we would conjecture, involves distinct changes in one's strategies or plans for linking the many motor acts to already existing sequential process-schemata in different 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 fall down 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 [illegible] should we say "only"?) has worked out for himself an unusually expressive abstract language for manipulating representations of physical activities.

4.1 Learning by Building Descriptions

[illegible] illustrate with [illegible] aspects of learning in the context of a procedure [illegible] by P. Winston to learn

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 them to find progressively larger structures and objects, and then attempts to "understand" them. It applies heuristic methods and the results of earlier learning experiments. Thus, the simple scene on the left is described by a network of abstract objects, relations, and relations between relations.



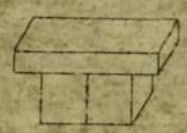
SCENE 1: AN ARCH

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.

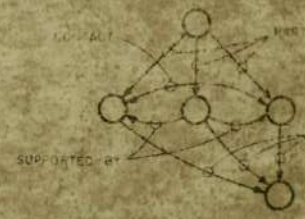
Note that since these properties describe only relative spatial relations, the very same network serves to describe both of these figures, which are visually quite different but geometrically the same.



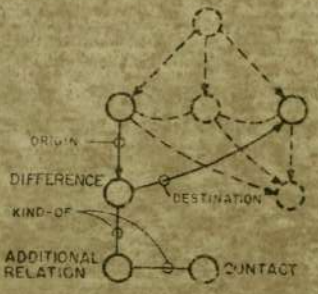
Next we present SCENE 3, to the left below, and the machine constructs the network shown to its right.



SCENE 2: NOT AN ARCH

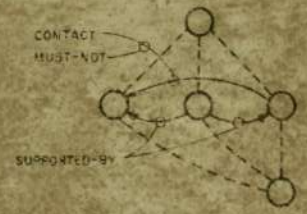


This differs from the network of SCENE 1 in only a few respects. If the program is asked what this structure "is", it will compare this description with others stored in its memory.



It has already networks for table, tower, and several other structures, but as in the expect, the structure it finds most similar to 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.

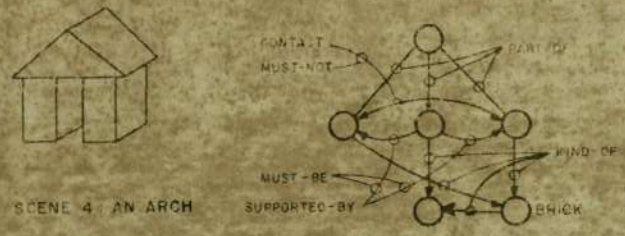


Now for the next example we present scene 4 and again this, too, is not an ARCH. The most prominent difference in this case is that the new structure lacks the support relations



SCENE 3 NOT AN ARCH

and the program for modifying "ARCH" now adds a "must-not-contact" pointer to the support relations. Finally, I present Winston's example, scene 4, and assert that this is an acceptable example of an ARCH.

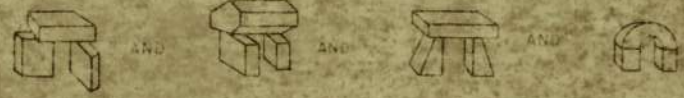


SCENE 4 AN ARCH

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

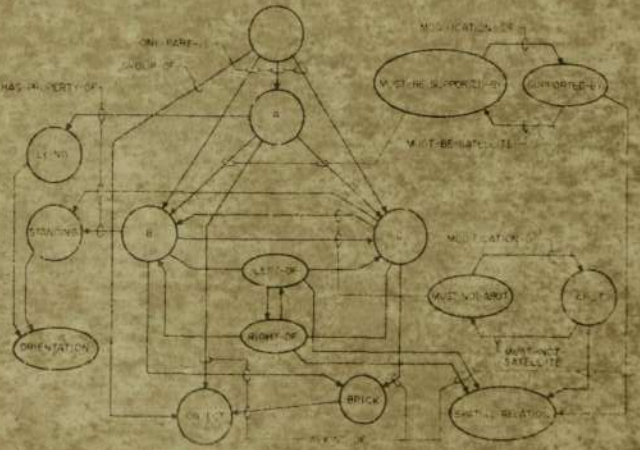
"A structure in which a prismatic body is supported by two upright blocks that do not touch each another."

We have just seen how the program learns to identify correctly the membership or lack of membership in the class ARCHES of two particular examples. In consequence, it will properly "understand" whether or not it should decide that



are also arches, because there are no "must-not" contact 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:



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 [illegible] differences between the two scenes [illegible]



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 graphic rotation of the scene — far (though 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 [illegible] step, but the ideas in the next section, on describing [illegible] 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:

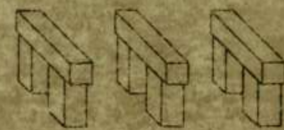
The ability to compare and contrast descriptions. (As we shall see in section 4.6)

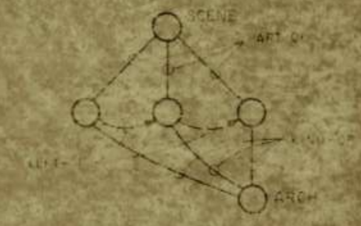
The ability to make deductions involving the concept, to adapt it to new situations.

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 [illegible] block scene following, before learning a concept of ARCH, it would have produced an impossibly complex and [illegible] 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 [illegible] aggregate [illegible] as much available [illegible] as it can apply. In doing this we encounter [illegible] problems very much like those we [illegible] in our sketch of Guzman's [illegible] program, and in many cases [illegible] strategies.

4.2 Learning by Being Taught

Imagine a child playing with a toy car and [illegible] to build an interesting structure to play with. If [illegible] Winston's program were present, he could learn [illegible] make an arch by the process just described, [illegible] 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 [illegible] for himself that is not too easy and not too hard — such as an obstacle that requires two hands to overcome, but cannot be [illegible] trivially with one hand.

In case (a) he knows how to tell which structures are in the class [illegible] while experimenting he will indeed find [illegible] too [illegible] the [illegible] variant of the [illegible] here we get the same [illegible] through the same mechanism — yet in humanistic terms the behavior would be described much more naturally [illegible] "play" or "undirected" activity [illegible] result [illegible] in structural terms [illegible]

4 OF 4
N72 25224 UNCLAS

"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 to [illegible] idea of an arch [illegible] it [illegible] purported [illegible] ... it would represent [illegible] formulation [illegible] acquisition of [illegible].

[illegible] we have [illegible] have [illegible] and [illegible] [illegible] the things that [illegible] [illegible] different.

The child [illegible] [illegible] have a decent chance of [illegible] structures.

To do this, [illegible] be able to describe [illegible] why an experiment is unsatisfactory. [illegible] his car between the supports, [illegible] of moving the supports apart. This [illegible] not [illegible] hard, since [illegible] against the obstacle will [illegible] this.

Since most experiments [illegible] structures [illegible] has to have some ability to reconstruct a usable version of earlier [illegible] situation after [illegible] disaster.

Without the teacher, [illegible] that he will [illegible] after just four trials! He must have enough persistence [illegible] [illegible] structure to carry through. To do this consistently would [illegible] of the problem's difficulty. Of course, if this is [illegible], he will find something else [illegible] not all play is productive!

Winston's program seems to [illegible] 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 [illegible] irreparably off the track, and he might have to back up [illegible] restart [illegible] [illegible] concept-learning [illegible] [illegible] that a teacher [illegible] [illegible] acquired [illegible] happened in the [illegible].

[illegible] important [illegible] the program [illegible] which [illegible] of the [illegible] explanations [illegible] that [illegible].

Thus, in building with blocks, the relations "support" and "contact" count to dominate properties of color, particular shapes and even other spatial relations like "in front of" or "to the right of."

[illegible]

4.6 Analogy, Again

We will return to our very first topic, solving problems by finding analogies, in Ch. 11 we proposed that the key problem is finding ways to adapt the clauses in descriptions. But this is exactly what happens in the programs we have just described. When asked to describe a new scene situation, Winston's program makes use of the other descriptions it remembers that it can describe the scene in terms of slightly [illegible] descriptions. Although we have not explained in detail how the [illegible] descriptions to contain that the result [illegible] descriptions, in this system, is itself a description! Actually, 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. The [illegible] descriptive elements have the same format as do the original scene descriptions. We can operate upon them with the same programs. In particular, two different description can be compared as readily as any other pair of descriptions.

Now we can apply this idea to the analogy problem. You 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:X]],

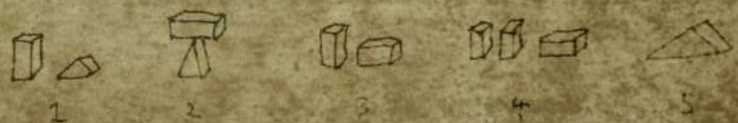
Diff[Diff[A:B] : Diff[C:X2]],

Diff[Diff[A:B] : Diff[C:X3]], etc.

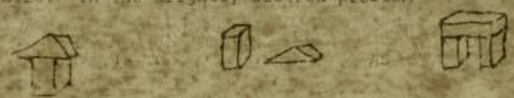
Each of these summarizes the discrepancies within the "analogical explanations" for each corresponding possible answer. So to make the decision, we have to choose the "best" or "simplest" of these. We will not give details of how this is done, it is described in Chapter 7 of Winston's thesis. But note 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





[illegible] chooses THREE as its answer, ONE as its second choice. In the slightly altered problem

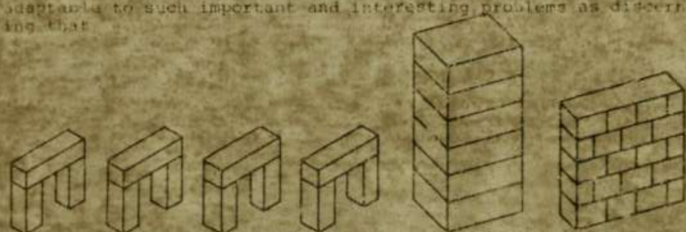


[illegible] 5 figures

It chooses FOUR as its answer.

4.7 [illegible] and Induction

The problem of recognizing or discerning groupings 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

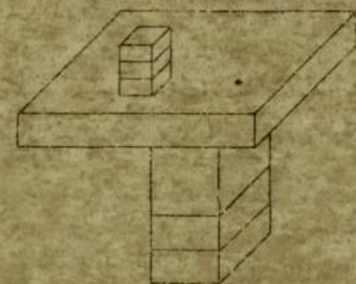


shows, not 12 + 6 + 20 = 38 objects, but "a row of arches, a tower of cubes, and a brick wall." More subtly, how do we "know" that one of these is three wedges while the other is three blocks? Visually, the lower objects in each tower are the same. These problems, too, can be treated by the same general 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 progresses best by [illegible] a very deep and thorough understanding of a few particular problems that [illegible] important [illegible] and insights, without which one [illegible] of speculations and generalities. [illegible] the present discussion in terms of [illegible] problems about induction and learning, the generality of the approach should speak for itself; we cannot [illegible] anyone believing that the usefulness of these ideas is in any important way confined to description of visual or mechanical structures!

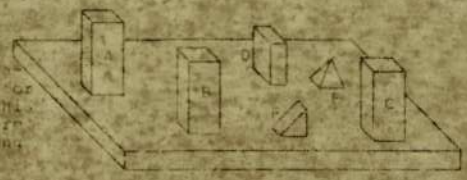
Take the groupings in the preceding figures and ask: "What qualities of the scene-descriptions characterize the intuitively acceptable groups." In some groups, like those shown above, it seems clear that the important feature is a CHAIN, say, of supported-by or in-front-of relations. In other cases it seems obvious that several objects show a common relationship to another. But no simple rules work in all [illegible].



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 process to separate out the three-block stacks in spite of the fact that the support-relation chain continues through all seven blocks.

We now 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.



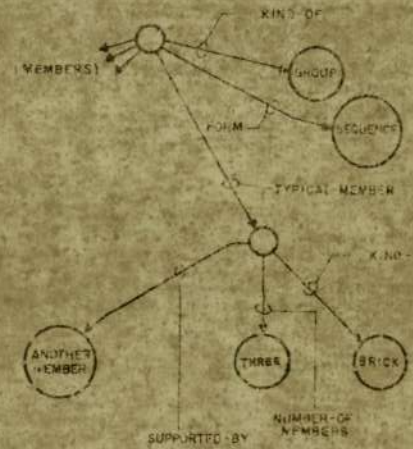
For example, [illegible] objects have the same [illegible] very nearly the same dimensions, they are [illegible] such [illegible] candidates for [illegible] group. The [illegible] on this [illegible] are [illegible]. All are polygons, all are standing, and all are supported by [illegible] board.

This program then examines [illegible] properties which [illegible] all, until a fairly homogeneous set remains. To do this, a program lists all relations [illegible] by [illegible] one of the candidates in the set.

When the procedure operates, the first pass through the loop rejects C and F, mainly on the basis of shape. (Size is not considered, in this pass, because the six objects are too [illegible] for "prism" to be put on the common-relations [illegible] 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. Finally, the procedure accepts only A, B and C into the group. Obviously this is appropriate for some goals, but not others.

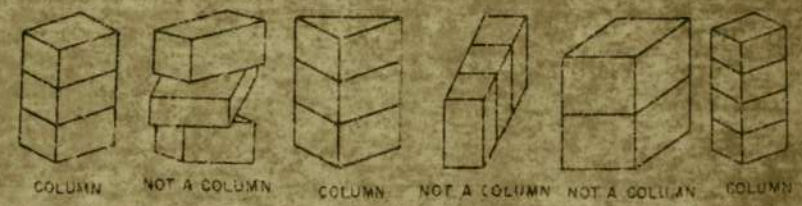
When grouping concepts are injected into the description framework, there can be many subtle and exciting consequences for other problems of induction. The figure on the next page shows the network representation obtained when the grouping process operates on the description of this 3-block column.





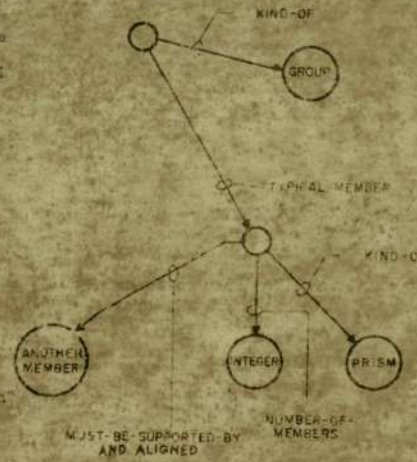
Into the description are introduced a "typical member" to which is attributed the common properties discovered by the grouping procedure. In this case, shape was used to form the group and the description includes the fact that there were three members. The [illegible] in.

In a learning experiment, the program is presented with the depicted sequence of scenes shown below and is told that the first, third, and fifth are instances of "column" while the others are not.



The second example causes the enforcement of a new pattern, "ALIGNED", a concept already available to the program that refers to the near 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 part of a group because it has only two elements, changes "one-part-must-be 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 attached 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.

5.6 Knowledge and Generality

We now turn to another set of questions connected with our long-range goal of understanding "general intelligence." An intelligent person, from a young child, is vastly more versatile than the "toy" programs we have described in Section IV. We can do many things reasonably well, and when one of our programs fails to do what it should, we may be able to change it, but this almost always requires major revisions and rethinking. An intelligent machine should be much more autonomous. It should often solve a new kind of problem by itself, or find how to proceed by asking for some kind of hint or help from a teacher.

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 create this procedure.

In any case, we think that much of what a person can do is picked up from his culture in various ways, and that 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 when even 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 means of 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, or 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 expressions 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.

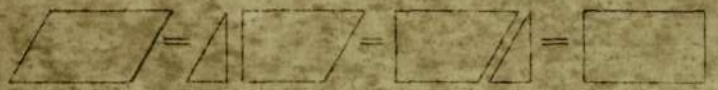
The KNOWLEDGE strategy sees progress as coming from better ways to express, recognize, and use diverse and particular forms of knowledge. This theory sees the problem as epistemological rather than as a matter of computational power or mathematical generality. It supposes, for example, that when a scientist solves a new problem, he engages a highly organized structure of specific, appropriate facts, heuristics, analogies, planning mechanisms, evaluation procedures, etc. To be sure, he also engages general problem-solving schemata but it is by no means obvious that these part could are that way directly because of the superior power of these general methods -- as compared with average people -- and 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 "thinkand" 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 new knowledge and the formation of new procedures, so 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 it is given 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.



Next, he studies this system near the latter "flag from that was uncovered by plan ULTRASIMPLE. It is [illegible] new model is probably very bad at other [illegible] which it will get false phenomena [illegible] something about relativity, he has reason to believe [illegible] earn [illegible] the frequency band [illegible] solves the new second-order equations, [illegible] pair of finite, close-together [illegible] plane. That "explains" the angular [illegible] model we abandon the "micro-world" and [illegible] slightly more sophisticated and better [illegible] understand situations. Our theories [illegible] and show are replaced by more specific [illegible] for studying he. Conflicting effect the fine-structure of the reasoner: a new plan, NONLINEAR or INELASTIC or THERMODYNAMICS or DISCRETE, or whatever his third-type knowledge needs is.

One cannot overemphasize the importance of this kind of scenario both in technology and in everyday thinking. We are absolutely dependent on having simple but adequate models of many phenomena. Each model -- or "micro-world" as we shall call it -- is very schematic in other words, it is [illegible] or second-order model. We talk about a universe in which things are so simplified that almost every statement about them would be literally false if applied directly to the real world. Nevertheless, we feel that they are important that we plan to devote a large portion of our effort to developing a collection of these micro-worlds and finding how to use 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.

6.2 Micro-worlds and Understanding

In order to study such problems, we would like to have collections of knowledge for several "micro-worlds", all ready to learn how to fit them together. Especially, we would like to take such a system and to extend its own knowledge base by understanding the kinds of information to be found in books. One might begin by studying the protocols of a student trying to understand the stories given to young children in schoolbooks. Any six-year-old understands much more about each of such topics as and various things as

time	space	planning	explaining
counting	going	recovering	allowing
failing	opening	attending	wanting
seeing	giving	promising	hurrying

[illegible] a typical student, as he will also [illegible] in developing the [illegible] and part of the following discussion is based on his experience.

In the next section we will describe a micro-world and subject matter has a close relation to the vision world already described. Its objects are geometric solids such as rectangular blocks, wedges, pyramids, and cylinders. They are moved and assembled into structures by a robot hand. Input is taken on the basis of deduction about such properties as shape, spatial relations, support, etc. These interact with a body of knowledge that is partly permanent and partly contingent on external commands and recent events.

5.1 Winograd's BLOCKS world

Note: Sections 5.1 through 5.6 are largely adapted from Terry Winograd's Thesis, for which I am responsible for the over-simplifications and misinterpretations.

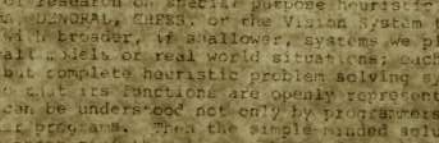
For developing and demonstrating his ideas about understanding natural language, Terry Winograd needed a micro-world in which to carry on a discourse containing statements, questions and commands. In this world we pretend we are talking to a very simple type of robot, like the ones being developed in 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 [illegible] ROBOT) (IS [illegible] HAND) (IS [illegible] PYRAMID)

For other, less basic properties we can write attribute-value statements like (MANIP :B5) and (PHYSOB [illegible]). 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 [illegible] can be treated as objects, so we can make such [illegible] 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)).

[illegible] 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 initial 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 [illegible]. An "antecedent theorem" can be regarded as a sort of demon that watches for some sort of event to happen, and then [illegible] 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 [illegible] 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 [illegible] between [illegible] [illegible]. We can give a name [illegible] [illegible] [illegible] BEHIND and [illegible] [illegible] the difference [illegible] [illegible] for X, Y and [illegible] [illegible] the transitive closure of SUPPORT. [illegible] supports B, B supports [illegible].

The [illegible] of [illegible] WIDTH and HEIGHT are [illegible] [illegible] assertions and [illegible] [illegible] [illegible] the [illegible] is used for comparisons like [illegible] [illegible] of the X, Y and Z coordinates [illegible] could [illegible] changed [illegible] in accord with human [illegible]. In order to compare measurements, we have the relation [illegible] sentence "B1 is shorter than B2" [illegible] [illegible] (MORE HEIGHT :B2 :B1).

OWN relates a person to any object. Knowledge about what the human user owns is gathered from his statements. [illegible] program can use statements about owning to generate [illegible] 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. [illegible] contains only [illegible] examples of such properties unrelated to the microworld of blocks.

[illegible] 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", [illegible] 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 [illegible] assertions that ought to be excised automatically when one of the participating objects is moved.

[illegible] of calling a consequent theorem to achieve a goal requiring motion, like (PUTON :B3 :B4), is a plan [illegible] 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:

(THGOAL (#MOVEHAND (600 200 300)) (THUSE TC-MOVEHAND))

nothing is actually moved. [illegible] "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 a goal [illegible], an [illegible] 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 the 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

[illegible] program's [illegible] questions about complex goals and [illegible] questions about its performance and about its intentions.

5.3.4 Carrying out Commands

Some theorems, like [illegible] GRASP [illegible] a series of actions. [illegible] definition of various [illegible] definitions, we will be able to follow [illegible] through a complex action in detail. [illegible] of activity [illegible] program can be found on the following page.)

```
(DEFEXPLANATION X
        GO      (COND ((GOAL (SUPPORT X) ON)
                       (GOAL (GET-RID-OF Y) (USE TC-GET-RID-OF))
                       (GO GO))
                      ((ASSERT (CLEARTOP X))))

(TC-GET-RID-OF X
        (OR
        (GOAL (PUTON Y (TABLE)(USE TC-PUTON))
        (GOAL (PUTON X Y)(USE TC-PUTON)))

(TC-GRASP X
        (GOAL (HAND) X)
        (WHEN (GOAL (GRASPING X))
              ((GOAL (GRASPING ?))
               (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 Y (TCENTER Y (SIZE X)))
        (GOAL (MOVEHAND Y) (USE TC-MOVEHAND))
        (GOAL (UNGRASP) (USE TC-UNGRASP))

(TC-PUTON X Y
        (PUT X Y)
        (GOAL (FINDSPACE Y (SIZE X) X)
              (USE TC-FINDSPACE TC-MAKESPACE))
        (GOAL (PUT X Y) (USE TC-PUT)))
```

Let us trace, for example, the expansion 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. Suppose that PLANNER will back up to reconsider the situation at the most recent previous variable-binding decision. For example, in this case the system may backup, for a place to put the block X, and stupidly decide to put it on Y. Now it must make another choice, and presumably this time it will be found to a different, more sensible location so that this time 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 SE (SIZE X) X)
      (USE TC-FINDSPACE TC-MAKESPACE))
```

which says to try to find a space on Y big enough for X, ignoring the space already occupied (possibly) by X. If location Y, resulting from a decision of this sort further down, is found to be a failure we would back up, but the program contains recommendations for how to find such a place. Specifically, it says to try to find a space already there, if this fails, then TC-MAKESPACE says to try to make such a space.

```
(GOAL (SUPPORT 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 recommended TC-GRASP checks to make sure the hand is empty, grasps the object by looking in the data base for (HAND ?), if the hand is already grasping the object it has nothing more to do, if not, it must first get the hand on the object. This may involve complications -- the hand may already be holding something so there may be an object sitting on top of the one it wants to grasp. In the first case it must get rid of whatever is in the hand. It does this by 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 and put it there, using the PUT and FINDSPACE, which perform all the necessary calculations, using information about the sizes and locations of all the objects. TC-PUT then creates a goal (PUT X Y) where the variable Y is bound to the location the hand must be moved to in order to put the object into the desired place, then calls MOVEHAND to actually put the object. If we look at the goal structure of our active goals at this point, assuming that we want to grasp B1, but we're already grasping B2, we see

```
(GRASP B1)
    (GET-RID-OF B2)
        (PUTON B2 TABLE)
            (PUT B2 (45) (0 0))
                (MOVEHAND (45) 10)
```

After having to put B2 on the table, and we have achieved the first part of the original goal of getting the hand over the block we want to grasp. TC-GRASP now

```
(GOAL (GRASP B1) (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 PUTC, 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, manipulately calling TC-GRASP again.

And so on. To answer questions about the past, the BLOCKS program remembers 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 STACKUP. 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 reasons for goals called by the linguistic part of the system are "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, in part, concerned with the problem of representing the meanings of a discourse from both the grammatical structure, and Winograd's theory about the interactions between meanings and syntactic structure are so intimate that it is a bit hard to try to draw a definite boundary here on the procedures that parse a sentence, the procedures that are to "understand" it (draw logical inferences about the things carried in the discourse,

Winograd's system is concerned primarily with how [illegible] several levels of detail and it is impossible to do justice to [illegible] summary. Our strategy will be first to present a dialog which illustrates its performance in the blocks world context, and then to discuss some of the mechanisms that are involved in producing this performance. In the following dialog, the system answers questions and answers questions about a simple scene containing several blocks and boxes, and selected attributes and properties of these objects are 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 responses. The other lines are comments explaining some of the features which are being exhibited.

The dialog was carried out in real time with the system manipulating 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 [illegible] lary 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 in the box.

The system has to figure out what is meant by "one" and "it", and also uses deduction to decide which block it is holding.

I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.
OK.

(Printed out to keep us informed.)
The hand has to set down the red block to pick up the taller blue one. Figure 2 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?

Pick up a big red block.

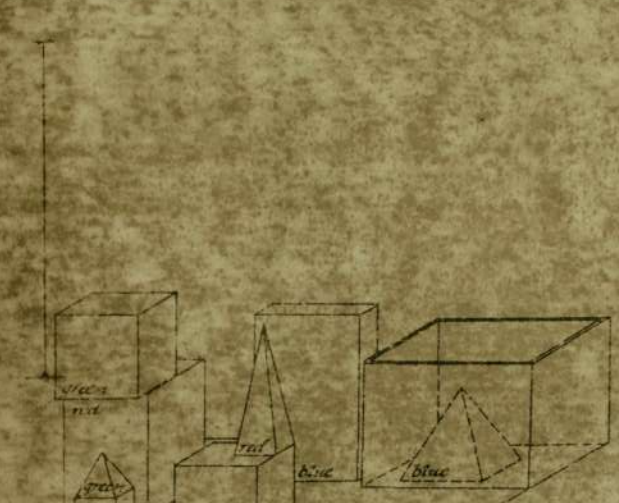


Fig. 1.

[illegible] many blocks are not in the box

[illegible] pyramids are supported

[illegible] I told you to pick up

[illegible] the phrase "at least one of them" in my instruction

[illegible] a question and the one which I just answered. There are also [illegible] which are complicated [illegible] such as "at least one", "more than three", "exactly seven", etc. The memory of past events is used [illegible] when [illegible] "the one" in my instruction I told you to pick up. The [illegible] the complicated grammatical structure in which "the one" is actually the object of a phrase embedded two levels deeper.

6. Is it supported? Passives can be used without a "by ..." phrase.
YES, BY THE BOX.

7. How many blocks did you pick up?
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 building it and examine the robot's program.

12. Pick up two pyramids.
I CAN'T
The robot tries and fails.

13. The blue pyramid is mine.
I UNDERSTAND.
This data is added to the data base.

14. I own blocks which are not red, but I don't own anything which supports a pyramid.
I UNDERSTAND.
The use of "own" is assumed to be the same as "owned" [illegible]

15. Do I own the box?
The box supports [illegible] pyramid.

...anything in the box?
YES, TWO THINGS: THE BLUE BLOCK AND THE BLUE PYRAMID.

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

...sentences such as "one", "it", "either", etc. are handled by 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 the 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 in figure 7.

...which cube is sitting on the table?
THE LARGE GREEN ONE WHICH SUPPORTS THE RED PYRAMID.
Okay, if the system would be satisfied with just calling this "a big green cube", 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 questions makes the dialogue very natural.

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

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

...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".

...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 one 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.

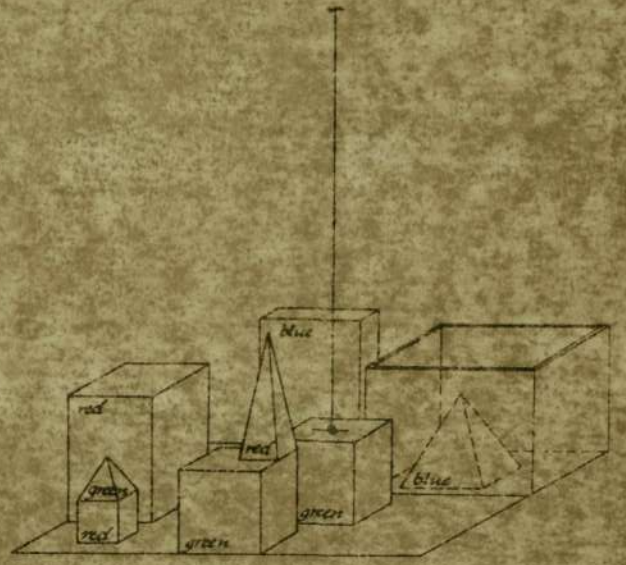


Fig. 7.

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 find facts about that event. Also in memory are complex descriptions of other events in terms of their relationships to each other. Pronoun reference must also fit with these descriptions. All of these are used in understanding phrases like "before you started to clean it off".

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 THAT CUBE; 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 strings 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 incomplete information about the basic properties of objects, including location, so it can use any statements about location or common sense rules to form beliefs, which it can then discuss or disagree.

34. put the blue pyramid on the block in the box.
This is syntactically ambiguous, but is understood without question 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 "STEEPLE"
The system interrupts immediately when it sees a word it can't figure out.

37. a "steeple" is a stack which contains two green cubes and a pyramid.
The quote marks indicate that we are defining a new word.

I UNDERSTAND.

35. Are there any steeples now?
BUILDER: NO.

39. Build one.

BUILDER: 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, the three large green cubes are 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 Large block "superblock".
BUILDER: OK. We can also give objects names.

41. Have you picked up superblock since we began?
BUILDER: YES.

42. Why did you drop it?
BUILDER: BY "IT", I ASSUME YOU MEAN THE SUPERBLOCK. BUILDER: 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?
BUILDER: YES, FOUR THINGS: THE BOX, THE RED CUBE, THE BLUE PYRAMID, AND THE BLOCK NAMED SUPERBLOCK.
When an object has a name, the name is used to help identify it in responses.

44 Thank you.
BUILDER: YOU'RE WELCOME.
Idioms can be defined without modifying the grammar.

7.5.1 Syntax and Meaning

What developments are involved in the blocks example? From the performance, it is clear that the system embodies a certain amount of knowledge about blocks, about building, about color, time, space, words, semantics, syntax, idioms, and common sense about questions, logical inference, instances of prior assertions, inconsistencies, failures to achieve goals, etc. Each such problem could perhaps be handled, in the particular situation, by appropriate tricks, special case heuristic 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 arise which were specifically not part of its design. In fact, perhaps, constructing theories involving meaning has always been put aside or postponed in favor of attempts to construct syntactic engines 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 the attempts here and more realistically comprehensive models have turned out to require a great body of special methods [illegible] systems that were unjustly hard to extend and finally incomprehensibly complex, just as was found true in the earlier approach. Perhaps, [illegible] attempt to split syntax [illegible] from semantics actually makes matters worse and both harder. Better by taking squarely the problem of meaning?

Such a proposal which once seemed such a difficult [illegible] syntactic analysis, now seems rather partly because it [illegible] turned out to be so difficult and partly because [illegible] heuristic programming made meaning so much less mysterious. It now appears that even a modest semantic component can greatly simplify understanding syntax.

To emphasize why pure 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 what "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 parses, but if we are interested in understanding how the language is used, we have to be able to make the choice. So in addition to 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 for understanding language, such attempts as were made to use such information took the form of pieces of fuller, patterns, and [illegible]. In Winograd's system, knowledge is expressed as procedures in special languages designed to give the flexibility and power of programs while retaining much of the clarity of the traditional, simpler rule formalisms. For example, knowledge can be a procedure, it can call on any other type of knowledge, when the parser can call arbitrary programs to see if the phrase it is looking for [illegible] and the syntactic [illegible] grammar [illegible] whether [illegible] a phrase makes [illegible] in the current [illegible] a conjunction [illegible] a preposition starting [illegible] the preceding [illegible] or on detailed [illegible] of the [illegible] text.

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.

As for the type of syntactic analysis he chose to use, he had developed a programming language [illegible] (PROGRAMMAR) that differs from other parsers in that the grammar is written in the form of a collection of programs. The grammar 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.

In between these is the semantic system which contains programs 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 5.5. 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 central problem: how does a sentence convey meaning beyond the [illegible]

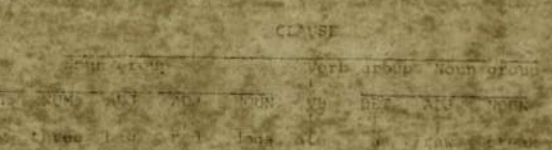
[illegible] structure and [illegible] general [illegible] [illegible] constructions [illegible] as a first approximation [illegible] remain.

The structure [illegible] series of grammatical [illegible] the [illegible] notion of generative grammar [illegible] of the syntactic [illegible] "feature" [illegible] the presence of [illegible] and [illegible] interpret [illegible] Of course, [illegible] "feature" [illegible] description [illegible] as simple lexicon [illegible]

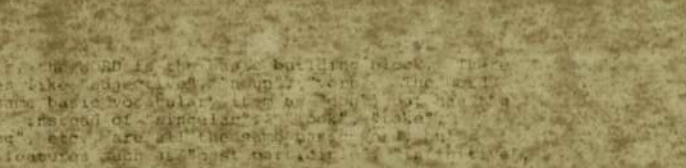
[illegible] system is based on a theory called Systemic Grammar (Halliday, 1967, 1970) which [illegible] features are primary. [illegible] of [illegible] on a "deep structure" tree, it describes the [illegible] features [illegible] depend on each other. In other forms of grammar, syntactic structures 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.



A not [illegible] exclusive [illegible] and IMPERATIVE is called a system and will be discussed in connection with certain [illegible] condition which can be an arbitrary [illegible] presence of other features. For example, in the grammar of the system that the feature MAJOR is the only one since only MAJOR clauses make the choice between DECLARATIVE, IMPERATIVE, and QUESTION. We can diagram some of our current features as:



The choice between SUBJ and MAJOR and the choice between YES-NO and WH- both depend directly on the presence of CLAUSE. Each type of relationship will be indicated by a pointer 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 two 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 did you show?".

In most current theories, there is no explicit [illegible] 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.

5.3 The Noun Group

We illustrate these ideas by presenting the structure of the NOUN-GROUP in some detail, [illegible] following the presentation in Winograd's thesis.

Here is the structure of the typical NG noun phrase that the same [illegible] can occur more than once. Most of those "slots" are optional, and may [illegible] in any particular NG.



The most important ingredient is the NOUN, which gives the basic information about the object or objects being referred to. Any NOUN immediately preceding the NOUN, there are an arbitrary number of "classifiers", like "plant life" or "water meter cover nut adjustment screw". The same class of words can serve as CLASF and NOUN in English, and our dictionary gives the ambiguity of words according to their word class, because words often have a special meaning when used as a CLASF.

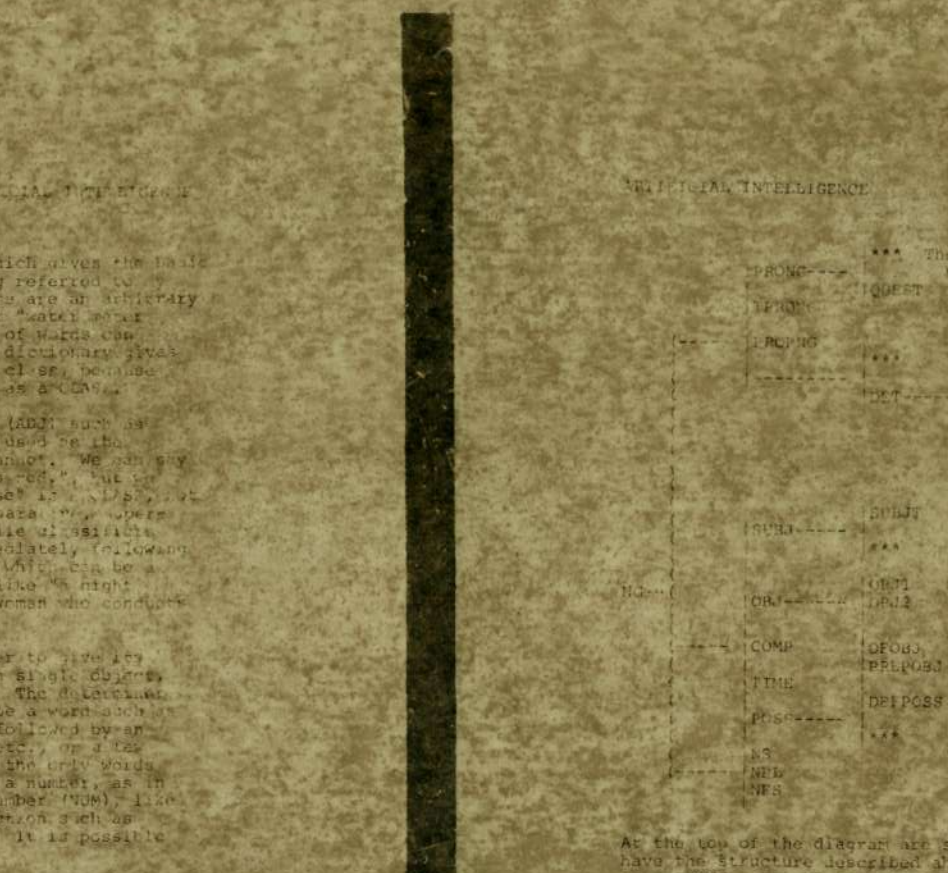
Preceding the classifiers we have adjectives (ADJ) such as "the beautiful soft red". Adjectives can be used as the complement of some CLASFR, but classifiers cannot. We can say "red hair", or "hair is red.", but we cannot say "That hair is wooden.", since 'horse' is a CLASF, not an ADJ. Adjectives can also take on the comparative and superlative forms ("big, bigger, and biggest"), while classifiers cannot ("horse, horsier, and horsiest"?). 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 Hell" or a CLAUSE REQ like "the woman who conducts the concert".

The first few elements in the NG work together to give its logical description -- that is 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 "this", 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", "3.5", or a more complex construction such as "all of those", or "more than a thousand". It is possible for a NG to have all seven 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 hand, let us look at the system network for NG.



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 TPRON marks a NG whose head is a specific 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 room". It has its own special syntax. The feature PROPNT 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 POSS to indicate this.

An INDEF NG can have a number as a determiner, "five cold frogs", or "at least a dozen eggs", in which case it has the feature NUMDET. Or it can be an INDEF determiner, such as "all". In

either case it has the choice of being a question. The question form of a NUMBER is "how many", while for other cases it is "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 and syntactically are between singular and plural, and between positive and nonnegative.

If a NG is either NUMB or QNTFR, it can be of a special type marked OF, as in "all of your trouble", but can also choose to be incomplete, leaving out the NOUN, as in "Give me three" or "I want some". There is a correspondence between the cases which can take the feature OF, and those which can be INCOM. No correct way either "two of them" or "Give me two". Possessives are an exception, we can say "Give me Juan's" but not "Juan's of them". And are handled separately.

The fourth 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, COMPLR, or object OBJ of various types. In addition, it can serve as the object of a PREPC (PPPSOBJ), in the role of "chocolate" in "a piece of chocolate". An NG can be marked as a TIME, as in: "Yesterday the world ended. On the day she left, all work stopped".

Finally, an NG can be the possessive determiner for another NG, as in "the cook's hatred" the NG "the cook" has the feature POSS, indicating that it is the determiner for the NG "the cook's hatred". Pronouns have the feature POSS.

When a PRONC 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 (labeled NPRO 0 S), 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 PRONC, there are special pronouns for first, second, and third person, singular and plural. The feature NPRO 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 kind 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 PREPC. If so, sees a preposition, it looks for an initial PREPC. If it sees a BINDER, it calls the CLAUSE program to look for a NOUN 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 tree 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 PP, the CLAUSE program marks the feature PRT on the verb 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 WHS clause (which would begin with a relative pronoun, like "back 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 OBJ1), asking the NG program to find an NG which will serve as an OBJ1. The structure is now:

```
(CLAUSE MAJOR IMPER PRT
   (VG IMPER
      (VB M'B 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 in 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 during the parse. The final result is:

```
(CLAUSE MAJOR IMPER PRT TRANS)
   (VG IMPER
      (VB M'B INF TRANS PRT)----------pick
      (PRT)---------------------------up
      (NG OBJ OBJ1 DET INDEF NS
         (DET INDEF NS)-----------------a
         (ADJ)--------------------------big
         (ADJ)--------------------------red
         (CLASF NS)---------------------block
```

The Semantic Structures

In 4.5 we describe 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 ideas. As one reads or reads linguistic sequences, one extracts meanings and uses them to modify one's model of the world 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 programs 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 objects, 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 with 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 decides that a certain structure, say a noun group, has been parsed. They look at both the syntactic structures and the meanings of the words (what properties represented by programs) and build up PLANNER theorems which can be used either by the deductive system as (for example) 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)
(EQUIV X)
(GOAL (COLOR X RED))
```

The variable "X" represents the object, and this description says that the object should be a BLOCK, it should have equal dimensions, and it should be red. A phrase such as "a red cube which supports the pyramid but is not contained in a box" would be built from the descriptions for the various objects, and would end up as:

```
(GOAL (IS X BLOCK)
(EQUIV X)
(GOAL (COLOR X RED))
(FIND 1 Y Z (GOAL (IS Y PYRAMID))
            (GOAL (SUPPORT X Y)))
(NOT (PROG (Z)
     (GOAL (IS Z BOX))
     (GOAL (CONTAIN Z X)))))
```

This "meaning" is a procedure. A larger deductive system could use it

to find such an object;

to see 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 contained a subsequent line (maybe itself, if which formally inaccurate expressions are sometimes permitted.

How does the semantic specialist 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 qualifiers and adjectives.

Part of the description for a noun uses semantic markers to restrict the meanings (interpretations) of a phrase. The BLOCKS world uses this type of semantic marker:



Again, vertical bars represent exclusive choices, while horizontal lines represent logical dependency. "PHYSOB" means "physical object", and "MANIP" means "manipulable object". The first specialist, SMAJ, finds that the definition of the noun "cube" is:

```
(NMEANS CUBE (IS X BLOCK) (EQUIV 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 LIST predicates (like EQUIV). It also can decide on recommendation lists to put onto the PLANNER goals, to guide deductions.

Next, SMAJ 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 comparation 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 noun. Here is the SMAJ definition for "big":

...program, it can then go on to examine the description (both the semantic markers and the PLANNER description) and produce an appropriate meaning relative to the object being described. This way, an absolute measurement (e.g., a "big elephant" is more than 1/10th mile) or a subjective (PLANNER definition of the form "the number of objects matching the description and smaller than the one being described is more than the number of suitable objects bigger than it is.")

In addition to the mapping of "red" to the semantic structures, the specialist must make choices in creating the PLANNER expressions. In the robot's tiny world, that wasn't much importance, but in the case of a phrase 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 which 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 probability 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))
      (GOAL X)----------------------PLANNER description
(BLOCK MANIP PHYSOB THING)--------semantic 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 planner 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 into 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))
      (GOAL X)
      (GOAL (IS X1 PYRAMID))
      (GOAL (SUPPORT X X1))----------PLANNER description
(BLOCK MANIP PHYSOB THING)----------semantics
(MANIP PHYSOB THING)----------------systems
(NS INDEF)--------------------------determiner
```


Relationships have the full capability to use semantic markers as objects do, and at an early stage of construction a relation structure contains a PLANNER description, markers, and systems in form identical to those for object structures (this is to share most of the programs, such as those which check for conflicts between markers). We can classify different types of concepts and relations (for example, those which are procedural, those which involve physical action, etc.) and use these markers to help filter out interpretations of clause modifiers. For example, the modifying phrase "without the hammer" in

"He left the house without a hammer."

has a different interpretation from that 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 get the correct interpretation.

TR-11 Sa[illegible]
[illegible]
March 1965 AD 612-7[illegible]

TR-[illegible] Samuel, Arthur L.
[illegible]
March 1966 AD [illegible]

TR-[illegible] Schorr, Allan Lee
[illegible]
[illegible] AD [illegible]

TR-13 Smoot, Francis John
A Heuristic Approach to Attribute
[illegible] Jet Such, S.B. & S.M.
[illegible]
Jan 1966 AD 474-018

TR-[illegible] Wambaugh, Joyce Elma
[illegible]
A Heuristic Computation and Analysis
[illegible] Thesis, Sloan School
September 15, 1965 AD 474-019

TR-[illegible] Denning, Peter James
Queueing Models for File Memory Operation,
S.M. Thesis, EE Dept.
October 1965 AD 624-948

TR-[illegible] Greenberger, Martin
[illegible]
Time Priority Problem
November 1965 AD 625 728

TR-21 Dennis, Jack B. and Earl C. Van Horn
Programming Semantics for Multiprogrammed
Computation
December 1965 AD 627-537

TR-[illegible] Rapkin, Mays Stephen Strong and John Hackett
MAP - A System for On-Line Mathematical
Analysis
January 1966 AD 476-411

TR-[illegible] Stratton, William Erwin
[illegible]
[illegible]
[illegible] S.M. Thesis, EE Dept.
March [illegible] AD 631-386

TR-[illegible] Ingalls, Daniel Huntley
[illegible]
[illegible]
[illegible]
March [illegible] AD 631-263

TR-27 Edwards, Daniel James
OCAS - On-line Cryptanalytic Aid System
S.M. Thesis EE Dept.
May 1966 AD [illegible]

TR-28 Smith, Arthur Ansel
Input/Output in Time-Shared, Multiprocessor [illegible]
of Dept.
June 1966 AD [illegible]

TR-29 Ryan, Sean Leon
Search Procedures Based on Measures of
Relatedness Between Documents, S.B.
Thesis, EE Dept.
June 1966 AD [illegible]

TR-30 Saltzer, Jerome Howard
Traffic Control in a Multiplexed Computer
System, Sc.D. Thesis, EE Dept.
July 1966 AD [illegible]

TR-31 Smith, Donald N.
Models and Data Structures for Digital
Logic Simulation, S.M. Thesis, EE [illegible]
August 1966 AD [illegible]

TR-32 Tuttleman, Warren
PILOT: A Step Toward Man-Computer
Symbiosis, Ph.D. Thesis, Math. Dept.
September 1966 AD [illegible]

TR-33 Pearson, Lewis M.
ANGPP - A Heuristic Program for Proving
Theorems of Group Theory, Ph.D. Thesis
Math. Dept.
October 1966 AD [illegible]

TR-34 Van Horn, Earl C.
Computer Design for Asynchronously
Reproducible Multiprocessing, Ph.D.
Thesis, EE Dept.
November 1966 AD [illegible]

TR-35 Reichel, Robert H.
An On-Line System for Algebraic Manipulation
Ph.D. Thesis, Appl. Math. Division
December 1966 AD 657-282

TR-36 Martin, William A.
Symbolic Mathematical Laboratory
Ph.D. Thesis, EE Dept.
January 1967 AD [illegible]

TR-37 Cozzin Arnold, Agnes
[illegible]
[illegible]
[illegible] S.M. Thesis, EE Dept.
February, 1967 [illegible]

APPENDIX A

TR-36 Greenfield, Ronald [by Daniel W. Lynch and Roger A. Humphray]
A Low-Cost Output Terminal for Time-Shared Computers
March 1967 AD 662-827

TR-37 Forte, Allen
Syntax-Based Analysis, Reading of Musical Notes
April 1967 AD 661-806

TR-40 Miller, James C.
On-Line Analysis for Social Scientists
May 1967 AD 668-09

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 Boyd-Canfield, Robert
On the Simulation of Dynamic Systems with Lumped Parameters and Time Displayed, S.M. Thesis, EE Dept.
October 1967 AD 661-672

TR-46 Hagen, Joseph W.
A Canonic Accelerator, S.B. Thesis, EE Dept.
November 1967 AD 663-393

TR-47 Moses, Joel
Symbolic Integration, Ph.D. Thesis, Math Dept.
December 1967 AD 662-868

TR-48 Yngve, Malcolm M.
Incremental Simulation on a Time-Shared Computer, Ph.D. Thesis, Sloan School of
January 1968 AD 672-325

TR-49 Iscoe, Fred L.
Asynchronous Computational Structures, Ph.D. Thesis, EE Dept.
February 1968 AD 672-802

TR-50 Denning, Peter J.
Resource Allocation in Multiprocess Computer Systems, Ph.D. Thesis, EE Dept.
May 1968 AD 676-231

TR-51 Marshak, Roger
CAEL, A Program which Solves Calculus Word Problems, S.M. Thesis, EE Dept.
July 1968 AD 67[illegible]

TR-52 Deitel, Harvey M.
Absentee Computations in a Multiple-Access Computer System, S.M. Thesis, EE Dept.
August 1968 AD 684-[illegible]

TR-53 Slutz, Donald R.
The Flow Graph Schemata Model of Parallel Computation, Ph.D. Thesis, EE Dept.
September 1968 AD 68[illegible]-511

TR-54 Groclow, 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 68[illegible]-4[illegible]

TR-55 Rappaport, Robert L.
Implementing Multi-Process Primitives in a Multiplexed Computer System, S.M. Thesis, EE Dept.
November 1968 AD 689-460

TR-56 Thornhill, D. E., R. W. Scott, D. T. Ross and J. E. Ward [ESL-R-356]
An Integrated Hardware-Software System for Computer Graphics in Time-Sharing
December 1968 AD 685-204

TR-57 Morris, James R.
An Efficient Method of Programming Languages, Ph.D. Thesis, Sloan School
December 1968 AD 683-334

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 Grimsdale, R. L. [illegible]
Computer Presentation of Three-Dimensional Objects in Visual Space, Ph.D. Thesis, EE Dept.
December 19[illegible] AD 692-[illegible]

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 689-965

TR-61 Baecker, Ronald M.
Interactive Computer-Mediated Animation, Ph.D. Thesis, EE Dept.
June 1969 AD 690-982

TR-62 Tillman, Roy [illegible]
EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation
June 1969 AD 692-962

TR-63 Brackett, John W., Michael Hammer, and Daniel L. Thornhill
Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for Use with a Storage-Tube Display Terminal
October 1969 AD 698-930

TR-64 Rodriguez, Jorge E. (Su-Shing)
A Graph Model for Parallel Computation, Sc.D. Thesis, EE Dept.
September 1969 AD 697-759

TR-65 DeRemer, Frank 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 699-362

TR-67 Vanderbilt, Dean H.
Controlled Information Sharing in a Computer Utility, Ph.D. Thesis, EE Dept.
October 1969 AD 699-591

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 10-011

TR-69 Getty, Geoffrey A.
Hierarchical Associative Memories, Parallel Computation, Ph.D. Thesis, EE Dept.
June 1970 AD 711-091

TR-70 [illegible]
Generalized Organization of Large Data Files: A Set-Theoretic Approach, [illegible] S.M. Thesis, EE Dept.
July 1970 AD [illegible]

TR-71 Prassinos, [illegible]
A Computer-Controlled [illegible] Processor, S.M. Thesis, EE Dept.
June 1970 AD 710[illegible]

TR-72 Faith, Rufus [illegible]
Coordination of Asynchronous [illegible] Ph.D. Thesis, EE Dept.
June 1970 AD 71[illegible]

TR-73 Griffith, Arnold K.
Computer Recognition of Prismatic Solids, Ph.D. Thesis, Math. Dept.
August 1970 AD 712-069

TR-74 Perlibakas, Edvard
Differential Geometric Approach to Integrated Circuit Design, Ph.D. Thesis, EE Dept.
August 1970 AD 712-070

TR-75 Hopper, Bruce
Symbolic-Address 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 F.
Optical Measurements for Language Recognition, S.M. Thesis, EE Dept.
October 1970 AD 715-194

TR-78 Hadrick, Stephen [illegible]
Design Strategies for File Systems, S.M. Thesis, EE Dept., Sloan School
October 1970 AD 714-267

TR-79 Horn, Berthold [illegible]
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-[illegible]

TR-80 [illegible] David D., Robert [illegible]
[illegible] Filter and [illegible]
[illegible] Theorem Information [illegible]
[illegible]
January 1971

APPENDIX

TECHNICAL REPORTS AVAILABLE

TR-91 Banks, Edwin R.
Information Processing and Transmission in Ideally Organized Acoustic Black Holes, Ph.D. Thesis, EE Dept.
January 1971 AD 717-951

TR-92 Krakauer, Lawrence J.
Computer Analysis of Visual Properties of Curved Objects, Ph.D. Thesis, EE Dept.
May 1971 AD 723-642

TR-93 Lewis, Thomas G.
On-Process Monitoring and Quality Control
Ph.D. Thesis, EE Dept.
January 1971 AD 717-950

TR-94 Wiederhold, Gio
Procedure Oriented Representations for Data in a Computer Program for Understanding Natural Language, Ph.D. Thesis, Math Dept.
August 1971 AD 731-399

TR-95 Miller, Perry L.
Automatic Creation of a Code Generator from a Machine Description, EE Degree, EE Dept.
May 1971 AD 724-731

TR-96 Schwarz, Jerome W.
Dynamic Reconfiguration in a Modular Computer System, Ph.D. Thesis, EE Dept.
June 1971 AD 725-859

TR-97 Thomas, Robert H.
A Model for Process Representation and Synthesis, Ph.D. Thesis, EE Dept.
June 1971 AD 727-049

TR-98 Welch, Terry A.
Bounds on Information Retrieval Efficiency in Static File Structures, Ph.D. Thesis, EE Dept.
June 1971 AD 725-124

TR-99 Jasny, Richard C., Jr.
Interactive Control Language Design
Ph.D. Thesis, Computer Sciences, Univ. of Wisconsin
May 1971 AD 728-036

TECHNICAL REPORTS AVAILABLE

TR-100 Jackson, James R.
Interactive Design Coordination for the Building Industry
June 1971 AD 728-100

TR-101 Ward, Philip R.
A Description and Flow Chart Language
Communications, Inc.
July 1971 AD 731-197

TR-102 Braun, Robert W.
A Resource Management and Control Monitor
(Formerly Programming Linguistics)
Tech. Rpt. No. 4, Dated 12, 1970
September 1970 AD 711-948

TR-103 Graham, Robert M.
Use of High Level Languages for System Programming
(Formerly Programming Linguistics)
Tech. Rpt. No. 5, November 30, 1969
September 1970 AD 711-86-

TR-104 Wulf, William A.
Suspension of Processes in a Multiprocessing Computer System
(Based on S.M. Thesis, EE Dept., February 1970)
September 1970 AD 713-989

TR-105 Balzer, Stephen N.
An Expansion of the Data Structuring Capabilities of PAL
October 1970 AD 720-761

TR-106 Bevard-Lawson, Gerard
Planned Random Sequences
(Based on S.M. Thesis, EE Dept., June 1970)
October 1970 AD 713-832

TR-107 Feldman, Jerome I.
Complexity Measures for Programming Languages
(Based on S.M. Thesis, EE Dept., September 1971)
September 1971 AD 729-011

TR-108 [illegible]

TR-109 Shapiro, Robert
A New List-Tracing Algorithm
October 1970 AD 7[illegible]

APPENDIX A

TM-20 Jones, Theodore
A Computer Model of Simple Forms of Learning
January 1968 AD 665-637

TM-21 Goldstein, Robert C.
The Substantive Use of Artificial Intelligence
April 1968 AD 674-618

TM-22 Wilks, Yorick
Situation, Inference and Belief
Machine Perception Systems and Its Environment
April 1968 AD 729-647

TM-23 Ornati, Alic. A.
The Relational Approach to the Management of Port Danger
April 1968 AD 721-615

TM-24 Goldstein, Robert C. and Albert V. Strnad
The MacIMS Data Management System
April 1968 AD 721-[illegible]

TM-25 Goldstein, Robert C.
Helping People Think
April 1968 AD 721-936

Project MAC Progress Report I
to July 1964 AD 465-089

Project MAC Progress Report II
July 1964-July 1965 AD 629-704

Project MAC Progress Report III
July 1965-July 1966 AD 648-716

Project MAC Progress Report IV
July 1966-July 1967 AD 681-842

Project MAC Progress Report V
July 1967-July 1968 AD 68[illegible]-870

Project MAC Progress Report VI
July 1968-July 1969 AD [illegible]-[illegible]

Project MAC Progress Report VII
July 1969-July 1970 AD [illegible]-[illegible]

* 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 prior than two years old $6.00, all others are $3.00 except TR-81 which is also $6.00.

Out-of-print may be obtained from MAC (see above).

All TM's have been deposited with DDC and are available only from NTIS using the AD number appended; the cost is $0.95 for microfiche and $3.00 for hard copy.

ARTIFICIAL INTELLIGENCE

AUTOMATA THEORY

CELLULAR AUTOMATA

COMBINATORIAL STRUCTURES

COMPUTER SYSTEMS RESEARCH

DYNAMIC MODELING, GRAPHICS AND NETWORKS

EDUCATION

IMPLICIT COMPUTATION

INTERACTIVE MANAGEMENT SYSTEMS

MATHLAB

PROGRAMMING LANGUAGES