University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS) Department of Computer & Information Science

June 1969

An Interactive Graph Theory System

Michael S. Wolfberg
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation

Michael S. Wolfberg, "An Interactive Graph Theory System", . June 1969.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-69-25.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/798
For more information, please contact repository@pobox.upenn.edu.

An Interactive Graph Theory System

Abstract

The medium of computer graphics provides a capability for dealing with pictures in man-machine communication. Graph Theory is used to model relationships which are represented by pictures and is therefore an appropriate discipline for the application of an interactive computer graphics system. Previous efforts to solve Graph Theoretic problems by computer have usually involved specialized programs written in a symbolic assembly language or algebraic compiler language.

In recent years, graphics equipment with processing power has been commercially available for use as a remote terminal to a large central computer. Although these terminals typically include a small general purpose computer, the potential of using one as programmable subsystem has received little attention.

These motivations have led to the design and implementation of an interactive graphics system for solving Graph Theoretic problems. The system operates on an IBM 7040 with a DEC-338 graphics terminal connected by voice-grade telephone line. To provide effective response times, computing power is appropriately divided between the two machines.

The remote computer graphics terminal is controlled by a special-purpose executive program. This executive includes an interpreter of a command language oriented towards the control of existence and display of graphs. Several interactive functions such as graph drawing and editing are available to a user through light button and pushbutton selection. These functions which are local to the terminal are programmed in a mixture of the terminal computer's machine language and the interpreted command language.

For more significant computational requirements the central computer is used, but response time for interactive operation is then diminished. In order to overcome the speed of the telephone link, the central computer may call upon a program at the terminal as a subroutine.

Based on the mathematical terminology used to define graphs, a high level language was developed for the specification of interactive algorithms. A growing library of these algorithms provides routines to aid in the construction and recognition of various types of graphs. Other routines are used for computing certain properties of graphs. Graphs may be transformed by some routines with respect to both connectivity and layout. Any number of graphs my be saved and later restored.

A programmer using the terminal as an alphanumeric console may call upon the programming features of the system to develop new interactive algorithms and add them to the library. Programs may also be created for the display terminal, using the central computer for assembly.

Examples of system use which are presented include finding a shortest path between any pair of vertices in a weighted directed graph, determining the maximally complete subgraphs of an arbitrary graph, interpreting a graph as a Mealy model of a finite state machine, and laying out a tree for aesthetic presentation.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-69-25.

This technical report is available at ScholarlyCommons: https://repository.upenn.edu/cis_reports/798

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
Philadelphia 19104

TECHNICAL REPORT

AN INTERACTIVE GRAPH THEORY SYSTEM

Prepared by

Michael S. Wolfberg

Moore School Report No. 69-25

for

Rome Air Development Center
Griffiss Air Force Base, Rome, New York

and

The Information Systems Branch
Office of Naval Research
Under
Contract NOnr 551(40)

June 1969

Reproduction in whole or in part is permitted for any use of the United States Government.

ABSTRACT

The medium of computer graphics provides a capability for dealing with pictures in man-machine communication. Graph Theory is used to model relationships which are represented by pictures and is therefore an appropriate discipline for the application of an interactive computer graphics system. Previous efforts to solve Graph Theoretic problems by computer have usually involved specialized programs written in a symbolic assembly language or algebraic compiler language.

In recent years, graphics equipment with processing power has been commercially available for use as a remote terminal to a large central computer. Although these terminals typically include a small general purpose computer, the potential of using one as a programmable subsystem has received little attention.

These motivations have led to the design and implementation of an interactive graphics system for solving Graph Theoretic problems. The system operates on an IBM 7040 with a DEC-338 graphics terminal connected by voice-grade telephone line. To provide effective response times, computing power is appropriately divided between the two machines.

The remote computer graphics terminal is controlled by a special-purpose executive program. This executive includes an interpreter of a command language oriented towards the control of existence and display of graphs. Several interactive functions such as graph drawing and editing are available to a user through light button and pushbotton selection. These functions which are local to the terminal are programmed in a mixture of the terminal computer's machine language and the interpreted command language.

For more significant computational requirements the central computer is used, but response time for interactive operation is then diminished. In order to overcome the speed of the telephone link, the central computer may call upon a program at the terminal as a subroutine.

Based on the mathematical terminology used to define graphs, a high level language was developed for the specification of interactive algorithms. A growing library of these algorithms provides routines to aid in the construction and recognition of various types of graphs. Other routines are used for computing certain properties of graphs. Graphs may be transformed by some routines with respect to both connectivity and layout. Any number of graphs may be saved and later restored.

A programmer using the terminal as an alphanumeric console may call upon the programming features of the system to develop new interactive algorithms and add them to the library. Programs may also be created for the display terminal, using the central computer for assembly.

Examples of system use which are presented include finding a shortest path between any pair of vertices in a weighted directed graph, determining the maximally complete subgraphs of an arbitrary graph, interpreting a graph as a Mealy model of a finite state machine, and laying out a tree for aesthetic presentation.

ACKNOWLEDGEMENTS

The author would like to express his appreciation to Professor Aravind K. Joshi as dissertation supervisor and to Professors John W. Carr III and Noah S. Prywes for acting as co-advisors especially in helping define this research activity. The author is also grateful to Professor Prywes as project supervisor who has made possible the environment for this work. For support and encouragement, the author is grateful to the Information Systems Branch of the Office of Naval Research and to Rome Air Development Center under Contract Nonr 551(40).

The work of many members of the MULTILIST Project has made this research possible. The author thanks Professor David K. Hsiao for providing the file system of the Problem Solving Facility and Dr. Richard P. Morton for the operating system. Also Dr. Morton has been instrumental in providing many valuable critical analyses and suggestions throughout the design stages. The author is indebted to Mr. Marvin Gelblat for his loyalty to the advancement and maintenance of the MULTILIST System in addition to his own research activities. The author thanks Messrs. William T. Park and David M. Kristol for their continued assistance in the development and maintenance of the PDP-8 and DEC-338 systems. Many ideas for implementation of DEC-338 input/output functions were suggested by Mr. Kristol. He has also altered and maintained the PSF program for the PDP-8, and he is the author of the CONSOL program for the DEC-338.

Thanks are due Mr. Thomas H. Johnson for his work on the PDPMAP Assembly System and Mr. Robb N. Russell for his help in the development of the XDDT debugging program.

The author would like to thank Messrs. Paul A. T. Wolfgang and William S. Mosteller for their help in IBM 7040 systems programming and maintenance.

Mr. Stanley Sobel deserves credit for becoming a user of the Interactive Graph Theory System and providing feedback on its ease of use. He has provided the "Miscellaneous Functions" monitor and the user programs INAM, XCGL, and FSAI.

The author would like to express his appreciation of excellent secretarial help from Miss Connie Murray.

As an unlimited source of confidence, encouragement, and patience, the author thanks his spouse who has also assisted in an editorial capacity and in various aspects of computer operations.

INDEX

algorithms 10, 19, 26-27, 212

ALL 92-99, 103

ALLA 22-30, 32, 69, 134, 212, 288

ALLHIT 88, 103, 122, 141

alter 19, 27, 94, 184-189

APL 22

arc 31, 61, 74, 83, 258

ARC 90-92, 94-99, 103

ARCCRN 123, 143, 161, 168

arrow 77, 95, 179, 184, 248

ARROW 90, 94-98, 103

ARROWD 91, 103

ASCII 89, 102, 104, 112, 129, 137, 147, 149-150, 166, 267

ATOI 166, 323

atom 30-31, 40-41, 48, 303

ATOM 48, 59, 168

bend 184

binary deck 26, 194, 279, 288, 322

BITS 165, 323

BITSIN 165, 323

blink 79, 85-86, 97, 155, 174, 179

BLINK 90, 97, 103

BLINKM 85, 103

BLNDIM 120, 141, 161, 168

block 24, 114, 123, 143, 250, 263, 302

buffer 115, 125-126, 128, 134, 151, 157, 163, 208, 321

CARD 61

change window 195-204

character 89, 102, 112, 116-117, 129-130, 149, 208, 211, 269

CHPSEU 88, 121, 141, 168

CHTBL 247, 269

CLRPB 88, 103, 110, 139, 168, 323

code conversion 164, 166
comment 60, 290
COMMON 60
communication cells 25, 69-70, 81-82, 107, 134, 140, 151, 274, 323
CONCOM 63
COORDS 90, 93, 100, 103
COPY 55
CRATOM 41, 59, 168, 293, 313
create 19, 40-41, 78, 91, 172-183
CREATE 40, 59-60, 168, 292-293
created internal names 77, 85, 88, 123, 143, 160
CRNAME 78, 91, 103
CRNAMS 161
CRNEIG 235-236
CRPAIR 41, 59, 168, 293
CRSET 41, 59, 168, 293
cursor 70, 80-81, 85, 87, 174
CURSOR 88, 103, 121, 141
data structure 20-21, 24, 28, 31, 288
Dataphone 12, 15, 105, 115, 134
DATSTR 310, 315, 323
declaration 32-34, 162, 164, 290
DECtape 14, 170, 244
DEC-338 7, 14, 16, 19, 27, 170
DEFALT 95, 103
delete 19, 46-47, 54, 78-79, 92, 190
DELETE 46, 59-60, 168, 292-293, 314, 318
DIM 90, 98, 103
DIMM 86, 103
dimming 79, 86, 98
directed 61, 66, 212
disk 7, 14, 16, 18, 25, 89, 114-115, 170, 244, 272
display list 252-257
DOG 128-129, 135-137, 166, 168-169, 274, 292, 294-295, 323
DOGDOG 323

DOGFLU 168, 296, 323
DOGFLUSH 158, 169, 292, 296
DOGGIE 19, 28, 67, 107, 247, 274
DOGI 208
DOGSET 139, 166, 168-169, 292, 294-295
DOGSTR 168, 296, 323
DOGSTRING 137-138, 169, 292, 295-296
DOGTBL 288
DOGTEX 168, 296, 323
DOGTEXT 137-138, 169, 292, 296
DREADY 115
DSPLAY 90, 96, 103
DTREE 66, 222

editing 27, 205
EIGHTH 73, 86, 103, 120, 141
element 31, 42, 46
empty 31, 40, 43, 49
EMPTY 49, 59, 168, 313
ENDDOG 131, 274
entity 8, 30-31, 303
ENTITY 32, 59, 168, 292
entity equality 47
entity expression 33
entity function 33-34, 38, 43
ENTITY FUNCTION 33, 39
entity property 33-34, 38, 58, 315
entity variable 30, 32-33, 38, 47
error 194, 290, 297, 319-320
ERROR 163-164, 168, 323-324
ESCAPE 152, 168-169, 195, 292, 297, 323
execution 19, 28, 134, 149, 192-195, 238, 244, 319
executive 9, 14, 18, 67
EXIST 90-95, 100, 103

field 89, 114

finite state acceptor 210-211

FORALL 49, 59, 168, 316-318

FORNXT 59, 168, 293, 316-318

FORTRAN 13, 18, 22-25, 27-30, 32, 37, 287, 297, 310

FOURTH 73, 86, 103, 120, 141

frame 196, 200-201

FRBLKS 123, 143, 168

free block 69, 250-252

from-vertex 31, 77, 162, 179

FSAI 210-211

FULL 73, 86, 103, 120, 141

FUNC 311

GDIM 161-162

GETDAT 85, 89, 108-109

GETGRA 168, 297, 323

GETGRAPH 158-159, 169, 292, 297, 321

GETSTA 168, 297, 323

GETSTATUS 151, 169, 292, 297, 320

GET12 160, 323

GMISC 161-162

GMON 109, 249

GOTO 89, 103, 105, 131, 134, 150, 157, 159, 321

GPACK 247

graph 8, 22, 31, 71, 82, 160-161, 194

graph input 158-162

Graph Monitor 19, 28, 109, 132, 152, 159, 163, 170-172, 279

Graph Theory 3-4, 22-23, 31, 244

GRAPHS 131-132

GRAPIN 159-162, 164-165, 168, 194, 323-326

GRUSER 279

GXWIND 161

GYWIND 161

G.SYS 322

INDEX (continued)

HALF 73, 186, 103, 120, 141
hardware 12, 16, 18, 247
HTIMER 123

IBM code 166-167
IBM 7040 12, 16, 18-19, 24, 27
IBSYS 13, 27, 194, 320, 323
INAM 209
INARC 61-62, 215
initialization 78, 85, 289
INNAME 136, 156, 161-162
INOUT 62, 323
INSERT 44, 59-60, 168, 292-293, 314
INTENS 120, 141, 161, 168
intensity 79, 85, 120, 141, 160, 257
interface 12, 15, 19
internal name 71, 74, 77, 160, 209, 258
interpreter 18, 28, 68, 84, 105, 107, 131
INTERS 235-236
INTRET 115-118
ITOA 137, 166, 323
IXSYS 319, 322

key 26, 190, 192, 194, 322
KRBKRB 112
KSFKSF 112

label 74, 77, 81, 95, 129, 174, 184, 267
LABEL 90, 95-98, 103, 168
LABEL1,LABEL2,... 161-162
LABEL(1),LABEL(2),... 144, 147, 149, 161-162
language 3-4, 8-9, 13, 18, 21, 23, 25, 27, 68, 71, 84, 105, 208, 288
LAYOTR 223, 226-228
layout 5, 80, 222-233
LELM 42, 59-60, 168, 291, 313
light button 19, 82, 109, 170, 172

light pen 6, 27, 67, 80, 140, 170, 196

light pen handler 81, 88, 122, 141, 151, 155, 270

light pen hit 67, 69, 81, 99, 122, 151, 155, 269

light pen pointing 80-81, 155, 270

light pen tracking 67, 69, 80-82, 88, 121, 141, 248, 270-272

LIST 90, 91, 103

LOAD 89-90, 103, 107, 115, 134

LOADGO 89-90, 103, 107, 134

logical-IF 60, 166

loop 77, 92, 94, 248

LOOP 94, 103

LOOPE, LOOPN, LOOPS, LOOPW 91, 95, 103

LPHIT1,LPHIT2,... 88, 122, 141, 155, 168

LTIMER 123

LTPEN 90, 99, 103

L^6 5, 24, 29, 245, 288, 302, 310

macro 13, 26, 28, 128-129, 274, 310

MANINT 110

manual interrupt button 109-110, 132, 150, 169-170, 172, 195, 281, 322

MAP 13, 24-25, 27, 29, 128, 160, 164, 274, 288-289, 310, 315, 323

maximally complete subgraph 229, 234-242

MCS 229, 234-242

MCS1 235-236

member 31, 43-44, 46-47, 49-50

MEMBER 49, 59, 168

memory structure 9, 20, 24, 27, 29, 302

MESSAG 138, 168

MINUS 95, 103

miscellaneous functions 206-207, 280

MKCMPL 299-301

modes 79, 85, 120, 260

Moore School Problem Solving Facility 8, 12, 18-19, 26-27, 170, 205, 319

MOVWIN 87, 102-103

INDEX (continued)

MULTILANG 27, 190, 192, 195, 205, 319
MULTILIST 12-13, 19, 26-27, 159, 190, 192, 194

NAMES 90, 94-98, 103
NDTREE 65, 222
neighbor 32, 156, 234-235
NEXT 99, 103, 235-236
NULL 48, 59, 168
NXTNEI 156

offset 74, 77, 95
OFFSET 95, 103, 161-162, 165
orientation 77
OUTARC 61-62, 215, 226

packing 165
pair 30-31, 40-43, 47-48, 303
PAIR 48, 59, 168
paper 67, 70-71, 73-74
paper tape 16, 18, 209-210
PB 144, 168, 323
PBCLR 111
PBS 144, 168
PBSET 111
PBSKIP 111
PDPMAP 16, 27-28, 69, 106, 128, 132, 153, 205, 247, 274, 282
PDP-8 12, 15-16, 20, 27-28, 170
PDP-8 Disk Monitor System 114, 170, 247, 249, 272, 280
PENPNT 100-101, 103
POP 46, 59, 168
POSWIN 86, 100, 103
PRBCD 53, 55, 59, 168
predicates 48
PRENT 53-54, 59, 168
preprocessor 23-25, 288
PRNAME 53-54, 58-59, 168
program segments 68, 89, 102, 115, 279

property 30-31, 34-36, 42, 46, 51-54, 315

PROPERTY 35, 162, 292, 316

property element 52, 306-308, 316

property name 30, 34, 37, 53, 55, 315-316

property set 52, 306-308

property type 30, 34, 54

PROPS 315-316, 323

PRSET 52, 59, 168

PRVAL 53, 58-59, 168

PSEUDO 87, 100, 103

pseudo-pen-point 70, 80-81, 85, 87, 100, 121, 141, 174

PUSH 45, 59-60, 168, 292-293, 314

pushbuttons 6, 15, 27, 68, 80-81, 88-89, 110-111, 132, 138, 140, 144, 151, 155, 169, 174, 238, 257, 265

pushdown 45, 226, 270, 294

RCVCH 115, 117-119

RELM 42, 59-60, 168, 291

remove 36, 44, 190-191

REMOVE 44, 59-60, 168, 292-293, 314, 318

REMPRO 59, 168, 292

REMPROP 36, 53, 60, 292

RESET 85, 103, 109, 250, 254

RESGET 109

restore 19, 27, 192-193

RESTRT 109

ring 25, 303

SAMPLE 169

save 19, 27, 190, 193, 205, 210, 244

screen 71, 73-74, 101, 222

SCREEN 100-101, 103

SELECT 154-155, 168, 282-287, 323

set 30-31, 40-41, 43-45, 47-48, 303

SET 48, 59, 168

SETCRN 88, 103

INDEX (continued)

SETCUR 87, 100, 103

SETINT 85, 103

SETPB 89, 103, 110, 139, 168, 323

SETWIN 86, 103

SETVAL 54, 59, 168, 291, 314 - 315

shape 74, 91, 93, 248

SHAPE 90, 93, 96-98, 103

shortest path 212-221

SHPATH 216-219

SHPTHW 213-214

SNDCH 115-117

START 88, 90-91, 93-100, 103

status 68-69, 82, 99, 120, 124, 140, 144, 150

STATUS 99, 103, 124, 144-145

STAT1,STAT2,... 124-128, 144-148, 161-162, 168

STLELM 59, 168, 291, 314

STOP 88, 90, 92-93, 95-99, 103

STRELM 59, 168, 291, 314

SUBR 311

subroutine 18, 29, 70, 106, 135, 152, 280, 311

SYSIO 114-115, 272-273

T 129, 131, 274

telephone 6, 12, 15, 18, 82

Teletype 12-13, 15, 27, 89, 112-114, 140, 149, 151, 164, 170, 208, 281, 319

TERMIN 168, 297, 323

TERMINATE 163-164, 169, 292, 297, 321

termination 150, 153, 155, 163-164, 281, 322

text 95, 102, 129, 137, 164

TEXT 95, 103

text console 8, 13, 15, 27, 195, 205

THROUGH 49-52, 56, 163, 292-293, 297, 314, 316-318

timer 123

TLSTLS 113

INDEX (continued)

to-vertex 31, 77, 162, 179

tree 65-66, 219, 222-233

TREELA 223-225

TSFTSF 113

TTYBUF 150, 168

TTYIN 149, 168

TYPE 89, 103, 105

UNDEF 30, 34, 48, 59, 168

UNDIR 62

undirected 62, 65, 229

unpacking 165

UNUSET 58-59

USEENT 57-59, 168

USEPR 58-59

USER 153, 280, 323

user message 132, 137-138, 154-156, 164, 245

user program 19, 26, 28-29, 69, 132, 152, 206, 274

use-set 56-57, 303, 307-308

USESET 56, 59

USETYP 57-59, 168

value 34, 53, 315

VERCRN 123, 143, 161, 168

vertex 31, 74, 82, 91-92, 258

VERTEX 90-93, 95-100, 103

WAITCH 168, 297, 323

WAITCHANGE 151, 169, 292, 297

WHOLE 90-92, 96-100, 103

window 19, 71, 73-74, 85-87, 120-121, 141, 160, 195-204

WINSIZ 120, 141, 161, 168

XCGL 209-210

XCOORD 161, 226

XOFF 144, 146, 149, 161-162, 168

XPSEUD 121, 142, 168

XWIND 121, 141, 168

YCOORD 161, 226

YOFF 144, 146, 148, 161-162, 168

YPSEUD 121, 141, 168

YWIND 121, 141, 168

TABLE OF CONTENTS

	page
TITLE PAGE	i
DEDICATION	ii
ACKNOWLEDGEMENTS	iii
INDEX	v
TABLE OF CONTENTS	xvi
LIST OF FIGURES	xxv
LIST OF TABLES	xxvii
BIBLIOGRAPHY	xxviii
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	3
1.1.1 Graph Theory	3
1.1.2 Computers and Graph Theory	4
1.1.3 Remote Computer Graphics Terminals	6
1.2 Objectives	7
1.3 Outline of the Dissertation	10
CHAPTER 2 SYSTEM DESIGN AND DEVELOPMENT	12
2.1 The Moore School Problem Solving Facility	12
2.2 Connection of the DEC-338	14
2.3 The PDPMAP Assembly System	16
2.4 A Display Executive	18
2.5 Role of the DEC-338	19
2.6 Data Structure	20
2.6.1 Division of Labor	20
2.6.2 Representation of Graphs	22
2.6.3 Implementation	23
2.6.3.1 Compilation	23
2.6.3.2 Memory Structure	24

		page
	2.7 Interaction	25
	2.8 Use of MULTILIST	26
	2.9 Summary of Capabilities	26
	2.9.1 What it Does	26
	2.9.2 How it is Done	27
CHAPTER 3	THE ALLA LANGUAGE	30
	3.1 Introduction	30
	3.2 Entity Variables	32
	3.3 Entity Functions	33
	3.4 Entity Expressions	33
	3.5 Properties	34
	3.5.1 Property Assignment	34
	3.5.2 Property Use	35
	3.5.3 Property Removal	36
	3.6 Entities and Properties - Some Examples	36
	3.7 Entity Creation	40
	3.7.1 Explicit Entity Creation	40
	3.7.2 Implicit Entity Creation	41
	3.8 An Atom	41
	3.9 A Pair and its Elements	42
	3.10 A Set and its Members	43
	3.10.1 Insert a Member	44
	3.10.2 Remove a Member	44
	3.11 A Set as a Pushdown	45
	3.11.1 Push	45
	3.11.2 Pop	46
	3.12 Entity Deletion	46
	3.13 Entity Equality	47

Page

3.14 Predicates 48

3.14.1 NULL 48
3.14.2 ATOM, PAIR, SET 48
3.14.3 EMPTY 49
3.14.4 MEMBER 49

3.15 THROUGH Statement 49

3.16 Property Handling 51

3.16.1 Property Set 52
3.16.2 Property Name 53
3.16.3 Property Value 53
3.16.4 Setting the Value of a Property 53
3.16.5 Property Type 54
3.16.6 BCD-Name of a Property 55
3.16.7 Illustrative Example: COPY 55

3.17 Uses of an Entity 56

3.17.1 Use-Set 56
3.17.2 Type of Use 57
3.17.3 Entity Where Used 57
3.17.4 Property Where Used 58
3.17.5 Illustrative Example 58

3.18 Miscellaneous Restrictions 59

3.18.1 Reserved Words 59
3.18.2 Statement Numbers 60
3.18.3 Logical-IF Statement 60
3.18.4 COMMON 60
3.18.5 Comments 60

3.19 Programming Examples 61

3.19.1 Cardinality of a Set 61
3.19.2 Incoming and Outgoing Arcs 61
3.19.3 Make a Graph Undirected 62
3.19.4 Connected Components of a Graph 62
3.19.5 Undirected Tree 65
3.19.6 Directed Tree 66

CHAPTER 4 DISPLAY OF GRAPHS GRAPHICAL INTERPRETIVE EXECUTIVE 67

4.1 Introduction 67

4.2 Interaction Through Communication Cells 69

Page

4.3 Organization 70

- 4.3.1 Paper, Window, and Screen 71
- 4.3.2 Vertices 74
- 4.3.3 Arcs 74
- 4.3.4 Created Internal Names 77
- 4.3.5 Existence and Display 78
- 4.3.6 Intensity, Blinking, Dimming 79
- 4.3.7 Light Pen 80
  - 4.3.7.1 Input and Tracking 80
  - 4.3.7.2 Pointing 81
- 4.3.8 Pushbuttons 81
- 4.3.9 Status of the Graph 82

4.4 DOGGIE Command Language 84

- 4.4.1 Miscellaneous Commands 85
  - 4.4.1.1 Initialization 85
  - 4.4.1.2 Intensity 85
  - 4.4.1.3 BLINK Mode 85
  - 4.4.1.4 DIM Mode 86
  - 4.4.1.5 Window Size 86
  - 4.4.1.6 Window Position 86
  - 4.4.1.7 Window Movement 87
  - 4.4.1.8 Pseudo-Pen-Point Position 87
  - 4.4.1.9 Cursor Position 87
  - 4.4.1.10 Light Pen Tracking 88
  - 4.4.1.11 Light Pen Hits 88
  - 4.4.1.12 Created Internal Names 88
  - 4.4.1.13 Pushbutton Clearing 88
  - 4.4.1.14 Pushbutton Setting 89
  - 4.4.1.15 Teletype Output 89
  - 4.4.1.16 Loading Program Segments 89
- 4.4.2 Graph Commands 90
  - 4.4.2.1 Creating a Vertex 91
  - 4.4.2.2 Creating an Arc 91
  - 4.4.2.3 Deleting a Vertex 92
  - 4.4.2.4 Deleting an Arc 92
  - 4.4.2.5 Altering a Vertex Position 93
  - 4.4.2.6 Altering a Vertex Shape 93
  - 4.4.2.7 Altering the Vertex Names of an Arc 94
  - 4.4.2.8 Creating an Arrow (and Altering Loop Orientation) 94
  - 4.4.2.9 Deleting an Arrow 95
  - 4.4.2.10 Creating and Altering a Label 95
  - 4.4.2.11 Deleting a Label 96
  - 4.4.2.12 Display Control 96
  - 4.4.2.13 Blinking 97
  - 4.4.2.14 Dimming 98
  - 4.4.2.15 Light Pen Status 99

Page

4.4.3 Graph Status 99
4.4.4 Coordinate Data 100

4.5 Encoding the DOGGIE Command Language 101

4.5.1 MOVWIN Command 102
4.5.2 Offset of a Label 102
4.5.3 Text Characters of a Label 102
4.5.4 Name of a Program Segment 102
4.5.5 TYPE Command 105

4.6 Interactive Programs in the DEC-338 105

4.6.1 Available Routines 107

4.6.1.1 The Interpreter 107
4.6.1.2 Restarting the Graph Monitor 109
4.6.1.3 Manual Interrupt Button 110
4.6.1.4 Pushbutton Handling 110

4.6.1.4.1 Clearing Pushbuttons 111
4.6.1.4.2 Setting Pushbuttons 111
4.6.1.4.3 Pushbutton Status 111

4.6.1.5 Teletype Input/Output 112

4.6.1.5.1 Teletype Input 112
4.6.1.5.2 Teletype Output 113

4.6.1.6 Using the Disk 114

4.6.1.6.1 Disk Activity Indicator 115

4.6.1.7 Dataphone Communications 115

4.6.1.7.1 Sending 116
4.6.1.7.2 Receiving 117

4.6.2 Status Information 120

4.6.2.1 Intensity 120
4.6.2.2 BLINK Mode and DIM Mode 120
4.6.2.3 Window Size 120
4.6.2.4 Window Position 121
4.6.2.5 Pseudo-Pen-Point Position 121
4.6.2.6 Tracking Indicator 121
4.6.2.7 Light Pen Handler 122
4.6.2.8 Created Internal Names 123
4.6.2.9 Available Storage 123
4.6.2.10 Timer 123
4.6.2.11 Graph Status 124

4.6.2.11.1 Vertex Status 124
4.6.2.11.2 Arc Status 126

4.6.3 DOGGIE Command Syntax 128
4.6.4 Sample User Program 132

Page

4.7 Interactive Programs in the IBM 7040 132

4.7.1 DOGGIE Commands 134
4.7.2 Extending the DOGGIE Language 137

4.7.2.1 User Messages 138
4.7.2.2 Clearing and Setting Pushbuttons 138

4.7.3 Values of DOGGIE Words 139
4.7.4 Status of the DEC-338 140

4.7.4.1 Intensity 141
4.7.4.2 BLINK Mode and DIM Mode 141
4.7.4.3 Window Size 141
4.7.4.4 Window Position 141
4.7.4.5 Pseudo-Pen-Point Position 141
4.7.4.6 Tracking Indicator 142
4.7.4.7 Light Pen Handler 142
4.7.4.8 Created Internal Names 143
4.7.4.9 Available Storage 143
4.7.4.10 Pushbuttons 144
4.7.4.11 Graph Status 144

4.7.4.11.1 Vertex Status 145
4.7.4.11.2 Arc Status 147

4.7.4.12 Teletype Input 149

4.7.5 Manual Interrupt Button 150
4.7.6 Requests for DEC-338 Status and Interaction 150

4.7.6.1 Executable Statements: GETSTATUS, WAITCHANGE, ESCAPE 151
4.7.6.2 Subroutines 152

4.7.6.2.1 USER 153
4.7.6.2.2 SELECT 154

4.7.6.2.2.1 Characteristics 154
4.7.6.2.2.2 Example 156

4.7.6.3 Buffering of DOGGIE Commands 157

4.7.7 Input of Graphs 158

4.7.7.1 Functional Description of GRAPIN 161

4.7.8 Termination 163

4.7.8.1 Normal 163
4.7.8.2 Error 163

4.7.9 Helpful Functions 164

4.7.9.1 Bit Handling 164

4.7.9.1.1 Unpacking 165
4.7.9.1.2 Packing 165

4.7.9.2 Code Conversion 166

Page

4.7.10 Reserved Words 166
4.7.11 Logical-IF Statement 166
4.7.12 Sample Interactive Program 169

CHAPTER 5 OPERATION OF THE TERMINAL 170

5.1 Graph Monitor 170

5.2 Create 172

5.3 Alter 184

5.4 Remove 190

5.5 Save 190

5.6 Restore 192

5.7 Execute 193

5.8 Change Window 195

5.9 Text Console 205

5.10 Miscellaneous Functions 206

5.10.1 DOGGIE Interpreter (DOGI) 208
5.10.2 Display Internal Names (INAM) 209
5.10.3 Paper Tape Storage of Graphs (XCGL) 209
5.10.4 Finite State Acceptor Interpreter (FSAI) 210

CHAPTER 6 APPLICATIONS 212

6.1 Introduction 212

6.2 Shortest Path 212

6.3 Tree Layout 219

6.4 Maximally Complete Subgraphs 229

CHAPTER 7 CONCLUSIONS 243

Page

APPENDIX 1 INTERNAL ORGANIZATION OF DOGGIE 247

A1.1 Introduction 247

A1.2 Minimum Hardware Requirements 247

A1.3 Storage Requirements 248

A1.4 Load, Start, Restart 249

A1.5 Storage Allocation 250

A1.6 Undefined DOGGIE Commands 252

A1.7 Display List 253

A1.8 Vertex and Arc Blocks 258

A1.8.1 Vertex Blocks 260
A1.8.2 Arc Blocks 263

A1.9 Label Blocks 267

A1.10 Light Pen Pointing 270

A1.11 Light Pen Tracking 270

A1.12 Use of the Disk 272

APPENDIX 2 USER PROGRAMS 274

A2.1 General Considerations 274

A2.2 Local User Programs 279

A2.3 User Programs as Subroutines 280

A2.4 SELECT Routine 282

APPENDIX 3 ALLA PREPROCESSOR 288

A3.1 Introduction 288

A3.2 Functional Description 289

A3.2.1 Declarations 292
A3.2.2 Data Structure Commands 292
A3.2.3 Control Statements 293
A3.2.4 DOG Statements 294
A3.2.5 Interactive Statements 296

	Page
A3.3 Error Messages	297
A3.4 An Example	299
APPENDIX 4 ALLA MEMORY STRUCTURE AND SUBROUTINE PACKAGE	302
A4.1 Introduction	302
A4.2 Memory Structure	302
A4.2.1 Rings	303
A4.2.2 Entity Blocks	303
A4.2.3 Property Sets	306
A4.2.4 Use-Sets	308
A4.3 Subroutine Package	310
A4.3.1 Form of Subroutines	310
A4.3.2 Examples	312
A4.3.2.1 LELM	313
A4.3.2 2 EMPTY	313
A4.3.2.3 CRATOM	313
A4.3.3 Maintaining Structure	314
A4.3.4 Properties	315
A4.3.5 THROUGH Loops	316
APPENDIX 5 INTERACTIVE EXECUTION	319
A5.1 Methods of Interaction	319
A5.2 The Basic System	322
A5.3 Listings of ERROR and GRAPIN	324

LIST OF FIGURES

			Page
Figure	2-1	Equipment Configuration	17
Figure	4-1	The Paper	72
	4-2	Vertex Shapes	75
	4-3	Display of Vertices and Arcs	76
Figure	5-1	Graph Monitor	171
	5-2	Create Options	173
	5-3	Create a Vertex	175
	5-4	First Vertex Created	176
	5-5	First Vertex Labeled	177
	5-6	More Vertices	178
	5-7	Ready to Create Arcs	180
	5-8	Creating First Arc	181
	5-9	First Arc Created	182
	5-10	More Arcs	183
	5-11	Alter Options	185
	5-12	Two Vertices Moved	186
	5-13	Two Moved Arcs	187
	5-14	A Bent Arc	188
	5-15	Label Alteration	189
	5-16	Parts Removed	191
	5-17	Change Window Options	197
	5-18	FOURTH Window	198
	5-19	Position the Window	199
	5-20	Smallest Window Frame Selected	200
	5-21	Window Frame Positioned	201

			Page
	5-22	EIGHTH Window as Chosen	202
	5-23	Paper to be Moved	203
	5-24	Paper Moved	204
	5-25	Miscellaneous Functions	207
Figure	6-1	A Weighted Graph	220
	6-2	A Shortest Path Computed	221
	6-3	A Tree	230
	6-4	The Same Tree After Layout	231
	6-5	The Same Tree with Arcs Permuted	232
	6-6	The Same Tree with Another Root	233
	6-7	An Undirected Graph	239
	6-8	An MCS Computed	240
	6-9	Another MCS Computed	241
	6-10	A Third MCS Computed	242
Figure	A1-1	A Free Block	251
	A1-2	Sample Display List Blocks	255
	A1-3	Safe Display File Alteration	256
	A1-4	Common Properties of Vertex Blocks and Arc Blocks	259
	A1-5	A Vertex Block	261
	A1-6	An Arc Block	264
	A1-7	Full Label Blocks	268
Figure	A4-1	A Ring with Three Links	304
	A4-2	Entity Blocks	305
	A4-3	Property Set of a Pair or Set	307
	A4-4	A Property Element Block	307
	A4-5	A Use-Set	307
	A4-6	Example	309

LIST OF TABLES

		Page
Table 4-1	Relationships Between Window and Paper	73
4-2	DOGGIE Words and Their Values	103
4-3	Trimmed ASCII Character Codes	104
4-4	Correspondence Between Characters and Character Terms	130
4-5	Correspondence Between Trimmed ASCII and IBM Character Codes	167
4-6	Reserved Words in Interactive ALLA Programs	168

BIBLIOGRAPHY

1. Ash, W.L., and Sibley, E.H.: TRAMP: An Interpretive Associative Processor with Deductive Capabilities, Proc. of the 23rd National Conference, Association for Computing Machinery. pp. 143-156, 1968.

2. Baecker, R.M.: Planar Representation of Complex Graphs. M.I.T. Lincoln Laboratory Technical Note No. 1967-1, Lexington, Massachusetts, February 1967.

3. Bartlett, W.S., et. al.: SIGHT, A Satellite Interactive Graphic Terminal, Proc. of the 23rd National Conference, Association for Computing Machinery. pp. 499-509, 1968.

4. Berge, C.: The Theory of Graphs and Its Applications. London: Methuen, 1964.

5. Bernholtz, A., and Bierstone, E.: Computer-Augmented Design, Design Quarterly 66/67. pp. 41-51.

6. Busacker, R.G., and Saaty, T.L.: Finite Graphs and Networks: An Introduction with Applications. New York: McGraw Hill, 1965.

7. Christensen, Carl, and Pinson, E.N.: Multi-Function Graphics for a Large Computer System, Proc. Fall Joint Computer Conference, 1967. pp. 697-711.

8. Christensen, Carlos: An Example of Directed Graphs in the AMBIT/G Programming Language, to be published in Proc. of the Symposium on Interactive Systems for Experimental Applied Mathematics. Washington, August 1967.

9. Clark, R.: The Use of Graph Theory in the Analysis of Spark Chamber Data, Proc. IEEE Transactions on Nuclear Science. pp. 108-112, August 1965.

10. Cooper, D.C.: Computer Programs and Graph Transformations. Carnegie Institute of Technology, Pittsburgh, Pennsylvania, September 1966.

11. Cotton, I.W., and Greatorex, F.S., Jr.: Data Structures and Techniques for Remote Computer Graphics, Proc. Fall Joint Computer Conference, 1968. pp. 533-544.

12. Digital Small Computer Handbook. Digital Equipment Corporation, Maynard, Massachusetts, 1968.

13. D'Imperio, M.: Data Structures and Their Representation in Storage: Part I, NSA Technical Journal. pp. 59-81, 1964 (Unclassified).

14. Dodd, G.G.: APL - A Language for Associative Data Handling in PL/I, Proc. Fall Joint Computer Conference, 1966. pp. 677-684.

15. Evans, D., and van Dam, A.: Data Structure Programming System, Proc. IFIP Congress 1968.

16. Freedman, H.: A Storage and Retrieval System for Real-Time Problem Solving. Moore School of Electrical Engineering Report No. 66-05, University of Pennsylvania, 1965.

17: Gorn, S.: Specification Languages for Mechanical Languages and Their Processors - A Baker's Dozen, Communications of the ACM. pp. 532-542, December 1961.

18. Gotlieb, C.C., and Corneil, D.G.: Algorithms for Finding a Fundamental Set of Cycles for an Undirected Linear Graph, Communications of the ACM. pp. 780-783, December 1967.

19. Gray, J.C.: Compound Data Structure for Computer Aided Design; A Survey, Proc. of the 22nd National Conference, Association for Computing Machinery. pp. 355-365, 1967.

20. Harary, F. (ed.): Graph Theory and Theoretical Physics. New York: Academic Press, 1967.

21. Harris, Z.: Mathematical Structures of Language. New York: Interscience Publishers, 1968.

22. Harrison, M.A.: Introduction to Switching and Automata Theory. New York: McGraw Hill, 1965.

23. Horwitz, L.P., et. al.: Index Register Allocation, Journal of the ACM. pp. 43-61, January 1966.

24. Hsiao, D.K.: A File System for a Problem Solving Facility. Ph.D. Dissertation, University of Pennsylvania, May 1968.

25. IBM 7040/7044 Operating System (16/32K): FORTRAN IV Language (Form C28-6329-3) and Macro Assembly Program (MAP) Language (Form C28-6335-2). IBM Systems Reference Library, IBM Corporation, New York.

26. Ingargiola, G.: A Representation Technique for Systems of Interacting Algorithms. Ph.D. Dissertation, University of Pennsylvania, 1967.

27. Johnson, T.H., and Wolfberg, M.S.: The PDPMAP Assembly System. Moore School of Electrical Engineering Report No. 68-11, University of Pennsylvania, October 1967. Also available as DECUS Program No. 8-166, DECUS Program Library Catalog (June 1968) and Addendum (March 1969). DECUS, Maynard, Massachusetts.

28. Joshi, A.K.: String Representation for Transformations. Transformations and Discourse Analysis Paper No. 58, University of Pennsylvania, 1965.

29. Joshi, A.K., and Hiz, D.: Transformational Decomposition, Proc. of the International Conference on Computational Linguistics. Grenoble, August 1967.

30. Joshi, A.K., Kosaraju, S., and Yamada, H.: String Adjunct Grammers. Transformations and Discourse Analysis Paper No. 75, University of Pennsylvania, 1968.

31. Karp, R.M., and Miller, R.E.: Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing, SIAM Journal of Applied Mathematics. pp. 1390-1411, November 1966.

32. Knowlton, K.C.: A Programmer's Description of L^6, Communications of the ACM. pp. 616-625, August 1966.

33. Kristol, D.M.: Modifications to CONSOL. Project MULTILIST Memorandum, Moore School of Electrical Engineering, University of Pennsylvania, May 1968.

34. Kristol, D.M.: Modifications to CONSOL(II). Project MULTILIST Memorandum, Moore School of Electrical Engineering, University of Pennsylvania, July 1968.

35. Kristol, D.M.: The DEC-338, with CONSOL, as a PSF Terminal. Project MULTILIST Memorandum, Moore School of Electrical Engineering, University of Pennsylvania, January 1968.

36. Kristol, D.M., and Gelblat, M.: PDPMAP Under the Morton/Prywes System. PDP-8/DEC-338 Users Memorandum, Moore School of Electrical Engineering, University of Pennsylvania, January 1969.

37. Kuo, S.S., and Young, W.K.: Computer Studies of the Traveling Salesman Problem. presented at the Sixth National Conference, Computer Society of Canada, Kingston, Ontario, June 1968.

38. Laurance, N.: A Compiler Language for Data Structures, Proc. of the 23rd National Conference, Association for Computing Machinery. pp. 387-394a, 1968.

39. Lefkovitz, D., et. al.: CIDS No. 5, Computer Programming for an Experimental Chemical Information and Data System. University of Pennsylvania, June 1968.

40. Lewin, M.H.: An Introduction to Computer Graphic Terminals, Proc. IEEE. Vol. 55, pp. 1544-1552, September 1967.

41. Machover, C.: Graphic CRT Terminals - Characteristics of Commercially Available Equipment, Proc. Fall Joint Computer Conference, 1967. pp. 149-159.

42. Mason, S.J., and Zimmerman, H.J.: Electronic Circuits, Signals, and Systems. New York: John Wiley, 1960.

43. Meetham, A.R.: Partial Isomorphisms in Graphs and Structural Similarities in Tree-Like Organic Molecules, Proc. IFIP Congress 1968.

44. Moore, E.F.: Shortest Path Through a Maze, Annals of the Computation Laboratory of Harvard University. Harvard University Press, Volume 30, 1959.

45. Morton, R.P.: On-Line Computing with a Hierarchy of Processors. Ph.D. Dissertation, University of Pennsylvania, December 1968.

46. Morton, R.P., and Wolfberg, M.S.: The Input/Output and Control System of the Moore School Problem Solving Facility. Moore School of Electrical Engineering Report No. 67-30, University of Pennsylvania, June 1967.

47. Myer, T.H., and Sutherland, I.E.: On the Design of Display Processors, Communications of the ACM. pp. 410-414, June 1968.

48. Newman, W.H.: A System for Interactive Graphical Programming, Proc. Spring Joint Computer Conference, 1968. pp. 47-54.

49. Ninke, W.H.: A Satellite Display Console System for a Multi-Access Central Computer, Proc. IFIP Congress 1968.

50. Ninke, W.H.: GRAPHIC 1 - A Remote Graphical Display Console System, Proc. Fall Joint Computer Conference, 1965. pp. 839-846.

51. Ore, O.: Theory of Graphs. Providence, Rhode Island: American Mathematical Society, 1962.

52. Paul, A., Jr.: Generation of Directed Trees, 2-Trees and Paths Without Duplication. Coordinated Science Laboratory Report No. R-241, University of Illinois, Urbana, Illinois, January 1965.

53. PDP-8/I Disk Monitor System. Digital Equipment Corporation Document No. DEC-D8-SDAA-D, Maynard, Massachusetts, April 1968.

54. Programmed Buffered Display 338 Programming Manual. Digital Equipment Corporation Document No. DEC-08-G61C-D, Maynard, Massachusetts, 1967.

55. Prywes, N.S.: Man-Computer Problem Solving with Multilist, Proc. of the IEEE. pp. 1788-1801, December 1966.

56. Richardson, F.K.: Graphical Specification of Computation. Department of Computer Science Report No. 257, University of Illinois, Urbana, Illinois, April 1968.

57. Roberts, L.G.: Graphical Communication and Control Languages, Second Congress of Information System Sciences. pp. 211-217, Washington: Spartan Books, 1964.

58. Robinson, J.A.: A Review of Automatic Theorem-Proving, Proc. of the Nineteenth Symposium in Applied Mathematics. pp. 1-18, Providence, Rhode Island: American Mathematical Society, 1967.

59. Rose, G.A.: Computer Graphics Communications Systems, Proc. IFIP Congress 1968.

60. Rovner, P.D., and Feldman, J.A.: The LEAP Language and Data Structure, Proc. IFIP Congress 1968.

61. Schurmann, A.: The Application of Graphs to the Analysis of Distribution of Loops in a Program, Information and Control. pp. 275-282, 1964.

62. Seshu, S., and Reed, M.B.: Linear Graphs and Electrical Networks. Reading, Massachusetts: Addison-Wesley, 1960.

63. Sibley, E.H., et. al.: Graphical Systems Communication: An Associative Memory Approach, Proc. Fall Joint Computer Conference, 1968. pp. 545-554.

64. Staudhammer, J., and Ash, M.: A Sufficiency Solution of the Traveling Salesman Problem. System Development Corporation Paper SP-2514/000/00, Santa Monica, California, August 1966.

65. Sussenguth, E.H., Jr.: A Graph Theoretic Algorithm for Matching Chemical Structures, Journal of Chemical Documentation. pp. 36-43, February 1965.

66. Sutherland, I.E.: Sketchpad: A Man-Machine Graphical Communication System. M.I.T. Lincoln Laboratory Technical Report No. 296, Lexington, Massachusetts, January 1963.

67. Sutherland, W.R.: On-Line Graphical Specification of Computer Procedures. M.I.T. Lincoln Laboratory Technical Report No. 405, Lexington, Massachusetts, May 1966.

68. Unger, S.H.: GIT - A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism, Communications of the ACM. pp. 26-34, January 1964.

69. Vaswani, P.K.T.: A Technique for Cluster Emphasis and Its Application to Automatic Indexing, Proc. IFIP Congress 1968.

70. Welch, J.T., Jr.: A Mechanical Analysis of the Cyclic Structure of Undirected Linear Graphs, Journal of the ACM. pp. 205-210, April 1966.

71. Wessler, B.D.: TracD, A Graphic Programming Language. Master's Thesis, M.I.T., Cambridge, Massachusetts, June 1967.

72. Wexelblat, R.L.: The Development and Mechanization of a Problem Solving Facility. Ph.D. Dissertation, University of Pennsylvania, December 1965.

73. Wolfberg, M.S.: Corrections to DEC-338 Manual (DEC-08-G61C-D). DEC-338 Users Memorandum, Moore School of Electrical Engineering, University of Pennsylvania, July 1967.

74. Wolfberg, M.S.: Determination of Maximally Complete Subgraphs. Moore School of Electrical Engineering Report No. 65-27, University of Pennsylvania, May 1965.

75. Wolfberg, M.S.: XOD-Extended Octal Debugging Program (September 1967), available as DECUS Program No. 8-89, DECUS Program Library Catalog (June 1968). DECUS, Maynard, Massachusetts.

76. Wolfberg, M.S., and Russell, R.N.: XDDT Extended Octal-Symbolic Debugging Program (April 1968), available as DECUS Program No. 8-127, DECUS Program Library Catalog (June 1968) and Addendum (March 1969). DECUS, Maynard, Massachusetts.

77. Wolfberg, M.S., and Wolfgang, P.A.T.: UP.L6 - An L^6 System for the IBM 7040. Internal Report, Moore School of Electrical Engineering, University of Pennsylvania, 1966.

78. 339 Graphics Software System Programming Manual (preliminary). Digital Equipment Corporation Document No. DEC-9A-XPDA-D, Maynard, Massachusetts, June 1968.

CHAPTER 1

INTRODUCTION

The medium of computer graphics provides a capability for dealing with pictures in man-machine communication. Graph Theory is used to model relationships which are represented by pictures and is therefore an appropriate discipline for the application of an interactive computer graphics system. Previous efforts to solve Graph Theoretic problems by computer have usually involved specialized programs written in a symbolic assembly language or algebraic compiler language.

In recent years, graphics equipment with processing power has been commercially available for use as a remote terminal to a large central computer. Although these terminals typically include a small general purpose computer, the potential of using one as a programmable subsystem has received little attention.

These motivations have led to the design and implementation of an interactive graphics system for solving Graph Theoretic problems. The system operates on an IBM 7040 with a DEC-338 graphics terminal connected by voice-grade telephone line. To provide effective response times, computing power is appropriately divided between the two machines.

The remote computer graphics terminal is controlled by a special-purpose executive program. This executive includes an interpreter of a command language oriented towards the control of existence and display of graphs. Several interactive functions such as graph drawing and

editing are available to a user through light button and pushbutton selection. These functions which are local to the terminal are programmed in a mixture of the terminal computer's machine language and the interpreted command language.

For more significant computational requirements the central computer is used, but response time for interactive operation is then diminished. In order to overcome the speed of the telephone link, the central computer may call upon a program at the terminal as a subroutine.

Based on the mathematical terminology used to define graphs, a high level language was developed for the specification of interactive algorithms. A growing library of these algorithms provides routines to aid in the construction and recognition of various types of graphs. Other routines are used for computing certain properties of graphs. Graphs may be transformed by some routines with respect to both connectivity and layout. Any number of graphs may be saved and later restored.

A programmer using the terminal as an alphanumeric console may call upon the programming features of the system to develop new interactive algorithms and add them to the library. Programs may also be created for the display terminal, using the central computer for assembly.

Examples of system use which are presented include finding a shortest path between any pair of vertices in a weighted directed graph, determining the maximally complete subgraphs of an arbitrary graph, interpreting a graph as a Mealy model of a finite state machine, and laying out a tree for aesthetic presentation.

1.1 Motivation

This dissertation is concerned with a research effort which is specific in its goal, but somewhat broader in its means of striving towards that goal. The title, "An Interactive Graph Theory System," indicates the goal, which is to place man and machine in a situation where effective solution of a class of problems may be performed. The interfacing vehicle is not only a graphical display, but a sophisticated terminal including a general-purpose computer. Thus, in addition to computer graphics, this research touches on the use of computer networks. The aim is not to advance the field of Graph Theory itself; the emphasis of the work described here is in the area of programming system design.

1.1.1 Graph Theory

Graph Theory is a branch of mathematics which is appreciated by many fields in a wide variety of applications. Electrical engineers often treat electrical networks, switching circuits, and communication networks as graphs.[6,20,42,62] Theoretical physicists apply Graph Theory to crystal structure [20] and high-energy physics [9]. Chemical structures can be considered as graphs, and therefore classification and matching techniques are suitable for graph-theoretical approaches. [39,43,65] Various kinds of linguistics analyses, in particular transformational analyses, are readily represented as graphs.[21,28,29,30] Graph Theory has a great impact on Computer and Information Science, especially in Automata Theory, Mechanical Languages, and Programming Languages.[17,22] Flowcharts are graphs, and so are the data and memory structures on which programming systems are based.[10,61] Some researchers have developed graph models of computation. [26,31] A graph-theoretic approach has been used in order to optimize index

register allocation in compiled programs. [23] Some approaches employ graphs as the very statements of a programming language. [8,48]

As Graph Theory is applied, the pure mathematicians continue to extend the field with theorems independent of application. The reader interested in learning about Graph Theory itself may refer to one of the popular texts. [4,51]

The point has been made that Graph Theory is being used as a framework in which to pose certain problems in order to benefit from the terminology, theorems, and related understanding. Perhaps this formalism appeals to many because of its vividly graphic nature where relationships are modeled as pictures. Therefore it follows that when computers are applied to graph-theoretic problems, it is of significant value to use the two-dimensional (or three-dimensional) medium of computer graphics.

1.1.2 Computers and Graph Theory

Graph Theory is applied to various aspects of computers, and computers likewise aid in solving some classes of Graph Theory problems. Much work of the Graph Theorist consists of creative mathematics such as theorem proving. Research in the computer field has attacked such problems [58], and this aspect would be a fundamental part of a computerized laboratory for studies in Graph Theory. Other aspects of the work are the development and application of algorithms, which are more susceptible to computer aid.

There have already been many special-purpose programs written to solve graph-theoretic problems, often in symbolic assembly languages or algebraic compiler languages. [5,18,37,43,52,64,65,68,69,70,74] These efforts have been carried out in batch processing environments where

no interaction could occur, particularly no graphical interaction. Having had experience in this field [74], the author has been motivated towards a high level language which can be used for graphs and the use of the medium of interactive graphics. In some stages during the development of an effective algorithm, the user can benefit from a monitoring feature which keeps him informed of the progress of computations. While not in itself interactive, such a feature would be a likely by-product of an interactive system. The interactive mode would be exemplified by a user's directing the course of an algorithm as he notices its attempts at seeking solutions in parts of a graph where there is no hope.

Although such motivations had been developing over the past few years, this research effort was sparked into action as a result of initial investigations by Ronald M. Baecker at M.I.T. Lincoln Laboratory during the summer of 1966. [2] His work was documented in a brief Technical Note where he described a specialized interactive graphics program for the TX-2 computer which aided in the studies of planar representations of complex graphs. The major significance of this paper, however, was Baecker's thoughts on the "use and design of a Graph Theory Package." Some of the design concepts employed in this research are to be found in Baecker's suggestions. There were, in fact, mutual influences since the author was in direct contact with Baecker at the time. For example, it is not by chance that Baecker has suggested L^6 as the language for maintaining graph structure.

Another research effort which was reported concurrently with the development of the Interactive Graph Theory System was carried out at Harvard University by William M. Newman. [48] On the PDP-1 Newman produced a system for interactive graphical programming using state

graphs as the form of programs. Although the programming system uses graphs as programs, it does not include the facilities for the solution of graph-theoretic problems. The most significant comparison of Newman's work to this work is the form of drawing the graphs. In Newman's system, all states are represented by circles to which and from which arcs are drawn, with arrowheads placed at the ends of arcs. This format tends to fill the screen quickly and when many arcs terminate on the same state, the abundance of arrowheads adds unnecessary clutter to the picture. In the Interactive Graph Theory System, one might expect, in general, that many vertices would be drawn on the screen. Therefore, vertices may be rather small dots, and in order to avoid clutter, arrowheads are positioned somewhere along the length of the arc.

1.1.3 Remote Computer Graphics Terminals

In addition to the application area of Graph Theory, this research includes system design for a remote computer graphics terminal. During the past four years, many such devices have been made commercially available [11,40,41,47,59], but there has been little experience acquired in the use of such hardware for useful processing at the terminal as well as in the central larger computer. One motivation has been to demonstrate how much computing may be done at the terminal.

The typical configuration being considered is a large computer, perhaps time-shared, with a voice-grade telephone link to a remote graphics terminal, over which typical transmission is at 2000 bits-per-second. The terminal consists of a general purpose computer augmented with special display processing hardware which drives a CRT of viewing area from 8 to 20 inches square. The terminal includes various input devices for the user such as keyboard, light pen, buttons, knobs, tablet,

etc. In addition to the core memory of the terminal's computer, there may be a disk or drum for programs and/or data. Prices of such terminals range from $75,000 to as high as $200,000 or more. At the lower end of the price scale is the Digital Equipment Corporation Programmed Buffered Display 338 (or DEC-338), which was used in this research. Other research groups have employed the same device, but their emphasis has been on doing nearly all computing in the central machine. Specifically, work has been carried out at M.I.T. Lincoln Laboratory, University of Michigan [63], and University of Illinois [56]. Bell Telephone Laboratory has done pioneering work [50] and is continuing to pursue this area.[3,7,49]

Depending on the duration and frequency of attention the central computer may give the graphics terminal, there is a tendency towards expanding the terminal into a powerful computation and storage facility. This depends upon the intended uses of the communication link. In a system design, it is necessary to weigh the hardware costs with the desired response times. For some applications the remote computer graphics terminal described above may not be adequate; an example of this is when a user must observe pictures on the screen at a dynamic rate of information exceeding that of the communication link.

The DEC-338 remote computer graphics terminal is suitable for incorporation into a system for Graph Theory, and thus it has been so used. Much can be done locally at the terminal without the central computer, but this is not the recommended approach for system growth.

1.2 Objectives

The overall objective of this research effort consists of the design and development of an experimental system which allows a researcher or student to interact on-line with a powerful computer through a remote

computer graphics display terminal in order to solve algorithmic problems of Graph Theory.

The above statement indicates the usefulness of the project, but the methods used in building such a system are also to be emphasized as objectives. At the beginning of the work, the hardware and software systems which were available to the author dictated an initial constraint. Since much of that environment had been previously designed and/or implemented by the author it served as an ideal experimental system which could be easily modified when necessary. Thus, an objective was to attach the DEC-338 into the multi-console system called The Moore School Problem Solving Facility in such a way that it could be used both as a text console and graphics terminal. A user would then have the ability of program development as well as applying programs already in the system. Furthermore, the Interactive Graph Theory system programmer would have the available tools to modify that system from the terminal.

A fundamental objective for the central computer was the development of a compiler-level programming language oriented towards graph-theoretic algorithms. In an attempt not to restrict the power of the language, its primitives would reflect a set-theoretic approach. Thus graphs may be defined in terms of entities which include atomic objects, ordered pairs, and sets. In addition, the language would allow for specifying arbitrary data associated with any entity, thus providing the power of modeling arbitrary data structures. As in many programming languages, other desirable features include flexible loop control and branching capabilities, use of subroutines and functions, ease of use, and ease of readability.

The language must be enriched with appropriate primitives in order that interactive programs may be written. Control of what is displayed is essential since automatic monitoring of an arbitrary structure is time-consuming and often undesirable.

Underlying the compiler-level language must be a collection of processors which maintain a memory structure. An objective is modularity so that changes may be made in the memory structure without affecting the primitives of the language which deal with data structure. Thus this system can be used as a framework for comparative studies of memory structures. For example, at this level, a doubly-linked list approach could be compared against a list with only forward pointers.

Out at the graphics terminal there must be an executive program which has control of the display and the small computer. It is to handle input/output functions as well as maintain a flexible display file organization and simple data structure capable of storing only those parameters relevant to the display of graphs. The executive program would therefore be special-purpose for this applications area in order to increase the potential for significant performance at the terminal.

The executive program should have a command language which is interpreted so that requests from the central computer may affect the data structure at the terminal. There should also be the ability to execute local interactive user programs at the terminal. Enough facilities should be available for a programmer to avoid detailed knowledge of the hardware of the terminal. As a result the programs tend to be more machine-independent.

Finally, there must be communications procedures in both the central computer and remote computer graphics terminal. In a system where

the terminal has computational power, there must be methods of altering the center of control. At certain times, control should be in the hands of the user, and thus at the terminal. At other times, the central computer must be able to direct the activities at the terminal.

There are other objectives underlying this system design which are common to many programming systems. These include storage and updating facilities for programs and data, modularity for easy modification of parts of the system, and available tools for system growth. Adequate response times are always desirable in an interactive system according to the user's notion of the complexity of requests. Of some concern in this experimental development is the efficient utilization of the resources.

The types of problems for which the system is being designed are mainly graph-theoretical algorithms, but it should also be a test bed for layout problems such as were considered by Baecker.[2] The development of applications packages based on graph-theoretic approaches is another direction in which the system may grow. Any problems where graphs are used are appropriate, but since the compiler language used must be somewhat general, it is expected that the system might be applied to other classes of problems as well.

1.3 Outline of the Dissertation

Chapter 2 presents the framework upon which the Interactive Graph Theory System has been built, and then proceeds to discuss the methods used in building the system. System organization is described, enhanced by discussions of the reasons for the ways the system operates both as a whole and in some detail.

Chapters 3 and 4 constitute the technical descriptions of the languages specific to this system. They may be used as manuals by programmers. Chapter 3 covers the compiler-level language which is used to process graphs. The chapter ends with some practical examples of graph-theoretic algorithms. Chapter 4 is concerned with the interactive and graphical aspects of the system. This includes a description of the executive program in the remote computer graphics terminal and the way in which it is used both locally and within the central computer as an enrichment of the compiler-level language.

Chapter 5 describes the operation of the terminal from the user's point of view, including some of the interactive operations which are performed locally.

An account of some of the practical problems which have been tackled by the system is presented in Chapter 6.

Chapter 7 concludes the body of the document with a critique of the work performed and suggestions for continued research.

The five appendices provide technical details underlying the computer implementation of the Interactive Graph Theory System. Since program listings of all parts of the system consist of nearly one foot of printout, they are not included in this document. Further information required by a maintainer of this system has been provided by the author in the form of memoranda.

CHAPTER 2

SYSTEM DESIGN AND DEVELOPMENT

This chapter begins by describing the background and environment used as a base for the construction of the Interactive Graph Theory System. The middle part presents the important building blocks of the system, and the final portion is a unification of the various aspects of the effort into a description of what the system does and how it operates.

2.1 The Moore School Problem Solving Facility

The MULTILIST Project has been a continuing research effort at the Moore School for many years.[55] It has been concerned primarily with techniques for information storage and retrieval (which is the source of the name), particularly with its relationships to problem solving. The project has focused on the design, development, and use of a hardware and software system called the Moore School Problem Solving Facility (MSPSF). The software is an attempt to combine storage and retrieval capabilities of a computer with its computational power to solve problems. The hardware used for this work consists of a multi-console system attached to an IBM 7040. Since the IBM 7040 did not directly support a variety of types of terminals, a PDP-5 was originally selected to serve as an intermediary or satellite of the IBM 7040. The PDP-5 has been since replaced by a PDP-8 which was almost program-compatible and much faster. The PDP-8 is attached to the IBM 7040 by a direct data connection which operates at memory speed. The PDP-8 services consoles over telephone lines. There are three ports for Teletype consoles, and a standard interface to a 201B Dataphone which operates over a 2400 b.p.s. private line. Originally, at the other end of the private line was a Bunker-Ramo Teleregister 200 series Universal Control Unit

with two attached consoles. Each console had one alphanumeric display, a keyboard, and a Teletype printer. The program written for the PDP-8 (named PSF) serviced the Teleregister consoles, Teletypes connected over Telephone lines, plus the on-line Teletype of the PDP-8. In addition to providing hardware interfacing, the PDP-8 interpreted a console control language used for program editing, job control, and output observation.

Although many consoles were attached to the IBM 7040, the central computer would only execute one job at a time in a "fast-batch" mode. Each console had an associated input file and output file allocated on the large disk of the IBM 7040. A user could perform editing functions on his input file or peruse his output at any time, but he would have to wait in a job queue to get processing time. When no console jobs were pending, normal jobs for the IBSYS Operating System could be run from the System Input Unit (either card reader or magnetic tape). The Input/Output and Control System of the MSPSF was described in a Technical Report.[46]

Initially, console jobs consisted only of use of the MULTILIST facilities, but later developments made available all of the IBSYS Operating System to console users.[45] This provided a multitude of programming languages, especially FORTRAN IV and MAP Assembly Language. In addition, a very useful feature was made available at the same time: macros at the level of the input to the operating system. In the MULTILIST environment, these macros may include statements which cause retrieval by description of BCD card images, binary card images, or other macros to be interpreted. Not only is this macro facility conceptually appealing, but it has been proven most useful.

The most recent improvement to the MSPSF system was the division of one all-inclusive file for storage into individual independent files. This work also included the facilities for file ownership, sharing, read-only access, etc.[24]

2.2 Connection of the DEC-338

As interest in computer graphics developed in the Moore School, the need for graphics hardware was inevitable. A DEC-338 Programmed Buffered Display was purchased for use by various projects of both students and research staff. This device includes a PDP-8 computer as one of two processors; the other is a specialized processor with its own operation codes oriented toward control of the attached CRT display. Both processors share a common 8K-word 12-bit memory with 1.5 μs cycle time. The display processor represents a lot of digital logic, and this is reflected back to the programmer so that it requires many weeks to master the programming of the DEC-338. This was a strong motivation for the use of an interpreted language for the display of graphs in the Interactive Graph Theory System. Since even the programmer writing interactive systems under the executive of this system need not know about the DEC-338 hardware, it is not further described here. The interested reader may refer to the manuals published by Digital Equipment Corporation[12,54] and a memo written by the author which clarifies some of the operations of the hardware not covered by the manufacturer literature.[73]

Although the DEC-338 can be used as a stand-alone system, it is configured as a remote computer graphics terminal. It has a small fixed-head disk of 32K 12-bit words, and one DECtape for more permanent storage.

It is equipped with a standard Dataphone interface to a 201B Dataset over private wires. This is the means by which the DEC-338 is attached to larger computers on the University campus. The telephone connection is used to attach this terminal into the MSPSF at the same spot where the Teleregister equipment tied in. Although the Teleregister Universal Control Unit had fixed message protocol, the DEC-338 could be programmed with any chosen design. In particular, it could be set up to mimic the actions of the Teleregister device. This was the first step taken in the software support of the DEC-338 in the MSPSF.[33,35] Since the Teleregister equipment supported two independent consoles, the "simulator" on the DEC-338 retained the flexibility of being considered either one of the two consoles at any one time. The DEC-338 on-line Teletype keyboard along with some of the pushbuttons replaced the Teleregister keyboard, and the Teletype printer was used for printed output. The display screen of the DEC-338 served the same function as the screen in the Teleregister console.

At this point no improvements had been made in the operational characteristics of the display consoles, but this initial step provided the necessary groundwork for upgrading the facilities to support the DEC-338 as a graphics terminal. In order to accommodate this requirement, some changes had to be made in message protocol between the intermediary PDP-8 and DEC-338. Also, code conversion which previously took place in the PDP-8 was eliminated so that messages sent to and received from the DEC-338 would be arbitrary binary sequences. Externally, the operation of the DEC-338 as a text console remained unchanged, but now the scene was set for the building of the Interactive Graph Theory System. Meanwhile, for other reasons, the Teleregister

consoles left the Moore School, and the hardware configuration of the MSPSF therefore has evolved to what is shown in Figure 2-1.

Next, the PSF program in the intermediary PDP-8 was modified, and additional programs were written in the IBM 7040 to support the transmission of binary output from the user's output file on the disk of the IBM 7040 to the DEC-338. This link was needed for sending the results of assemblies of PDP-8 assembly language programs to the DEC-338 for either storage on the disk or punching out on paper tapes.[34] The assembly system is described in the next section.

2.3 The PDPMAP Assembly System

The PDP-8 (or DEC-338) is a small computer, but it can be programmed to perform many valuable tasks. One task for which many feel it is inappropriate is the assembly of PDP-8 (or DEC-338) programs. Thus the author instigated the development of a powerful assembly system for the small computer which runs on the IBM 7040.[27] This PDPMAP Assembly System includes extensive macro facilities, literals, location counters, line printer listings with cross-referencing, and a flexible cross-page linking facility. It has played a vital role in the development of the DEC-338 programs in the Interactive Graph Theory System.

PDPMAP has also been used by the author and others for the development of systems programs for the PDP-8 and DEC-338 which have lead to more effective use of the small computers. Two programs in particular were developed in order to provide adequate debugging aids needed during the development of the DEC-338 portion of the Interactive Graph Theory System.[75,76]

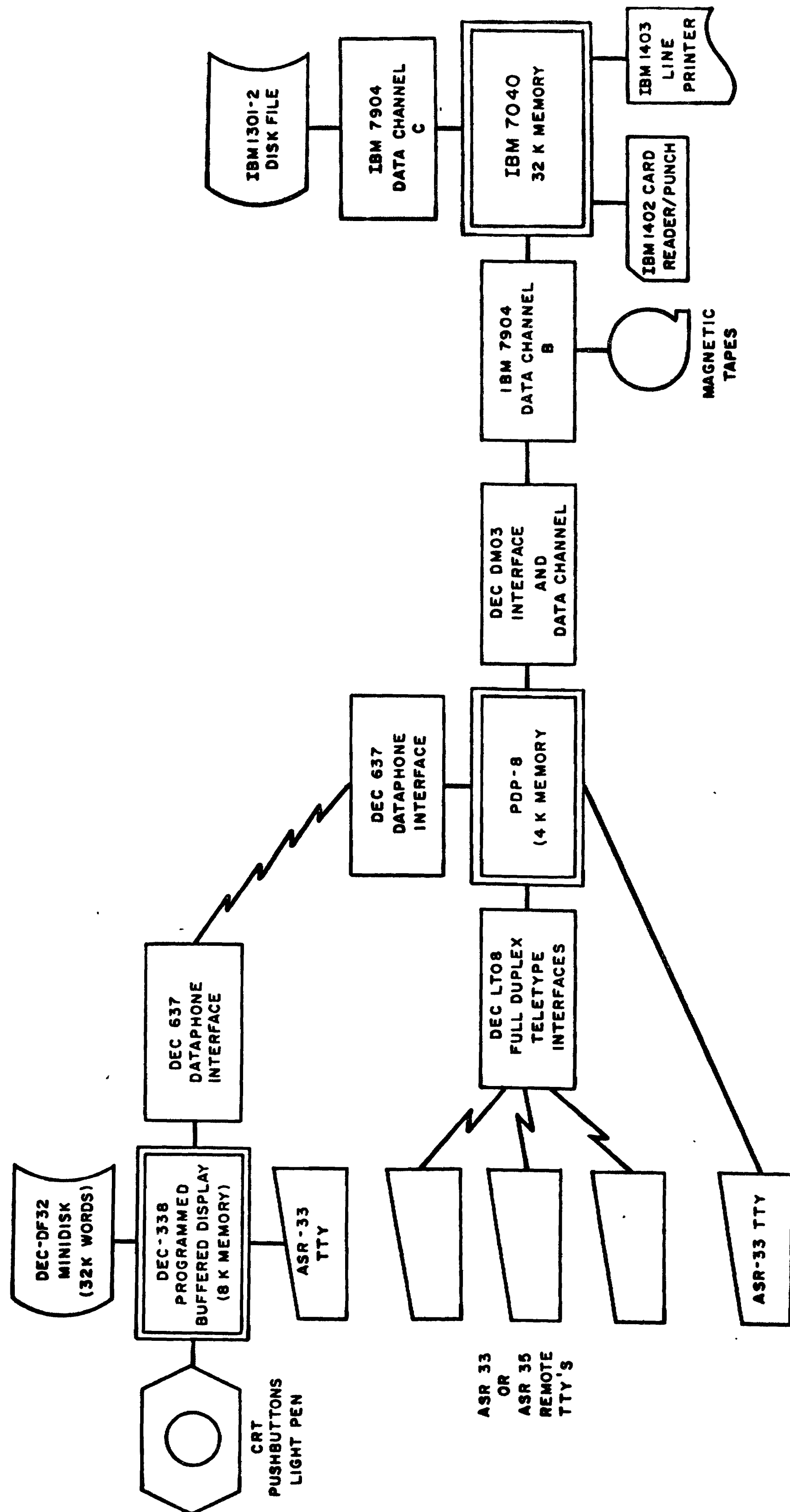


FIGURE 2-1 EQUIPMENT CONFIGURATION

In the MSPSF, PDPMAP can be used from the DEC-338, and the binary output from an assembly may be transmitted to the remote terminal for either storage on the disk or punching out of paper tapes.[34]

2.4 A Display Executive

The first design problem which was confronted at the outset of this research was a solution to the problem of displaying a graph on the screen of the DEC-338. Changes may occur in such a picture and so more than a display file must exist. As the various operations which would be desirable in the display of graphs are formulated, it is natural to list such operations as commands such as "move vertex V_i to X_j,Y_j." One possible approach would be to make available several subroutines which could be called with arguments. This approach is often used in driving a display within a FORTRAN environment (see, for example, the display support for the DEC-339[78]). An alternative approach was adopted which involved the design of a language to be interpreted within the DEC-338. The motivation here was based on the transmission of such commands by the central computer across the telephone line to the remote terminal. The language approach also seemed appropriate for affecting the display of graphs locally at the terminal. Thus any changes which could be made in the displayed graph would be done through the interpreter. By using only this special language, it is a relatively easy task to program the display since the underlying executive program provides all of the hardware-dependent programming. Another advantage of this unified method is the possibility of substituting other interpreters so that the language could direct other graphical devices such as a plotter or microfilm recorder.

The language which was designed for the control of the display of graphs also includes the primitives which reflect the notion of a graph existing in the DEC-338 without being displayed. Chapter 4 describes the role of the DEC-338 in this system as controlled by the Display of Graphs Graphical Interpretive Executive (DOGGIE).

2.5 Role of the DEC-338

It has already been indicated in previous sections that the DEC-338 plays two distinct roles in the Interactive Graph Theory System. First, it can be an alphanumeric display console of the MSPSF. Its second mode of use makes the terminal into a graphic display console for the handling of graphs. In this mode the underlying display executive is in control of the DEC-338, and it is considered the most fundamental or primitive level of the basic system. On the next level, interactive graphical programs may be implemented entirely within the DEC-338. The author has provided an extensive set of such programs thus making the terminal into an effective interface to a user. A "Graph Monitor" presents light buttons which can be used to control the creation, alteration, and deletion of graphs. The user is given the power to view a graph through four window sizes, and there are controls to alter the window position. Thus a user of the system has no need to know the underlying system structure, and there is no mention of this structure during normal use of the terminal. Three of the functions available to a user call upon the IBM 7040 for processing. Graphs may be saved by description in the MULTILIST file, and later restoration by description of any number of graphs may be performed. Interactive execution of previously compiled algorithms on the IBM 7040 (in conjunction with the DEC-338) gives the user the computing power unattainable in the terminal

computer alone. Chapter 5 describes the operation of the terminal under the Graph Monitor.

Some of the programs which support the available functions of the Graph Monitor are written in a form which may be used by new local programs developed by system programmers. The most useful of this class are those groups of subroutines responsible for communications with the IBM 7040 (via the PDP-8).

2.6 Data Structure

It has been suggested that the term "data structure" has been too widely applied.[13] There are two distinct issues which should be explicitly distinguished: the data elements and the relationships among them vs. the utilization of the physical storage devices with respect to bits, pointers, etc. The term "data structure" will be used to describe the former issue, and "memory structure" for the latter. The word "structure" will be used to encompass both concepts.

2.6.1 Division of Labor

A central concern of many computer graphics systems is the technique used in modeling the relationships among various data. This is of concern for the underlying implementation of the problem being attacked as well as for the maintenance of the display for the purposes of interpretation of user inputs, alteration of the picture, etc. When one central processor is used for the implementation of a computer graphics system, the tendency is to use one structure which encompasses both levels of information as has been done on the TX-2 computer at Lincoln Laboratory.[57,66,67] When the graphical work is done at a terminal which includes a computer, the organization heavily depends on the rate of information transfer. In the Graphic-I organization,

nearly all structure appeared in the central computer since the remote computer could be jammed with a display file at very high speed. [50] In a more divided system, where a voice-grade line links the terminal to the central computer, at least a display-oriented data structure must be maintained at the terminal for effective interaction.

One solution to the need for two structures in two (often) different computers is the implementation of the same basic structure in both machines. Possibly, one machine might be able to handle a subset of what the other can do. This type of approach has been used in the ASP-1 and ASP-7 implementations [19], and was a motivation in the development of TRAC-D. [71] A strong reason for taking such an approach is the possibility of writing some programs which may operate in either machine or both machines. Also, the systems programmer may be dedicated to using only one language.

Although the above type of approach is logically pleasant, unless the central and terminal computers are appropriately related, it could be stifling to the effectiveness of both machines. The structure in the smaller computer might be so general that its small memory is too quickly exceeded, and, at the same time, the possibility for sophistication in the larger machine might be suppressed. The division of labor being used in the Interactive Graph Theory System is based upon the problem area and the particular machines involved, especially the smaller one. Since it is an operational system, its effectiveness can be demonstrated. The author feels the approach is successful.

The data structure used in the DEC-338 is primitive with just enough features to handle the existence and display of graphs. The details of the data structure are presented in Chapter 4, and the under-

lying memory structure is covered in Appendix 1.

2.6.2 Representation of Graphs

An important part of the design of the Interactive Graph Theory System is the data structure and memory structure used in the IBM 7040. The approach taken has been influenced by the experimental orientation of university research so that sufficient modularity has been kept for other students to use this system as a test bed for further studies. Instead of providing a data structure capable of representing graphs only, a more general approach was selected which provided an environment where graphs could be represented as they are defined in Graph Theory textbooks. Thus the constructs of set theory are used as the primitives of the data structure. The need for data associations has prompted the inclusion of the facility for associating an arbitrary amount of data with any element of the data structure. Other systems which include this type of power are APL (Associative Programming Language) [14], LEAP [60], TRAMP (Time-Shared Relational Associative Memory Program) [1], and work involving list processing in the MAD compiler language[38].

The data structure used in this research is based on the APL approach. APL was "designed to be imbedded in PL/I as an aid to the user dealing with data structures in which associations are expressed." The IBM 7040 implementation of FORTRAN IV was extended in much the same way in creating the associative language ALLA. Some of the primitives of ALLA are taken from APL, but the operations appear as an extension of FORTRAN rather than PL/I. One deficiency of APL which has been eliminated in ALLA is the necessity for specifying in advance the allowable associations an entity may have. For use in graph-theoretic constructs, the ability to dynamically associate new types of data is important when

extending the graph structure to include new attributes such as "color of a vertex" or "neighbor." Chapter 3 is devoted to the ALLA language.

2.6.3 Implementation

The primitives of ALLA were selected on the basis of need to express certain relationships commonly used in Graph Theory and in an attempt to provide full structure-scanning facilities for an arbitrary given structure. The data structure design consists of the syntax and semantics of ALLA presented in Chapter 3. An issue which is somewhat independent is the underlying implementation of the compilation of ALLA statements and the way in which the memory structure is maintained.

2.6.3.1 Compilation

Since ALLA is an extension of FORTRAN IV, compilation must consist of at least that of the FORTRAN language. Therefore, it is rather advantageous to use the already-existing compiler as part of the process. There are two obvious possibilities: either the compiler may be changed to process ALLA statements, or ALLA statements may be changed to be processed by the compiler. In the given environment, the second alternative is the easier to accomplish. For reasons of legibility it was decided not to make ALLA statements conform exactly to the syntax of FORTRAN. The only method, therefore, was to preprocess ALLA into FORTRAN IV, and then finish the compilation by applying the FORTRAN compiler to the output of the preprocessor. Although this discussion has presented the design process as sequential, it is the case that the ease of preprocessing entered into the choice of the ALLA syntax.

The process of transforming ALLA into FORTRAN IV is one of string manipulation. In selecting a language in which to write the preprocessor, assembly language was avoided in order to attempt machine-independence.

The SNOBOL language was available on the IBM 7040, but it was unreliable and had no way to output text which the FORTRAN IV compiler could later process. Since L^6 was already being used (see the next subsection), that language was chosen. Appendix 3 describes the preprocessor, giving the transformations it performs on ALLA programs.

2.6.3.2 Memory Structure

The data structure and associative components of the ALLA language are preprocessed and compiled into FORTRAN SUBROUTINE and FUNCTION calling sequences. Up to that point, the underlying implementation is not reflected; it could be written in any language capable of being linked with FORTRAN and the memory structure could be anything. The method selected for the implementation of the memory structure is not coding in assembly language, but in order to retain some machine-independence and to make it easier to write, debug, and modify, the language L^6 was chosen. The language was originally designed and implemented at the Bell Telephone Laboratories where it received the name "Bell Telephone Laboratories' Low-Level Linked List Language" or L^6 (pronounced "L-six").[32] Based on the original implementation on the IBM 7094 using BELL MACRO-FAP, the author implemented UP.L6 for the IBM 7040.[77] In the process of translation, improvements and new features were added thus making it easy to link L^6 programs with both FORTRAN and MAP assembly language.

The memory structure underlying ALLA is written in L^6 which imposes only the organization of dynamic memory into blocks of 2^n (where n ranges from 0 to 7) full words. There is no preset linkage such as found in CORAL[57]; instead L^6 is a language in which the CORAL memory structure could be implemented. In fact, the ALLA memory structure which has been implemented is based on the same general

organization of data as doubly-linked rings. The ALLA memory structure is described in Appendix 4.

2.7 Interaction

The preceding sections have introduced some of the subsystems of the Interactive Graph Theory System, but they were not tied together. The role of the DEC-338 as a display terminal has been explained as being directed by an interpreted command language. Local programs are written using that language, and the important link joining the central and terminal computers is also by means of the same language.

The ALLA language was introduced as an extension of FORTRAN IV which provides data structure and associative processing. That language has been enriched once more by adding the primitives for interaction with the remote computer graphics terminal. This level of the language is called "Interactive ALLA," and that is actually what the preprocessor is capable of transforming (see Appendix 3). Interactive ALLA includes statements for sequencing of control and determining the activities of the terminal through standard status communication cells. Furthermore, interactive ALLA programs may include the symbolic form of the command language which drives the terminal. These capabilities are developed in detail in Chapter 4.

The preprocessor transforms interactive statements into FORTRAN SUBROUTINE calls upon a package of subroutines written in MAP assembly language. These routines effect communication with the DEC-338 via the input file and output file on the disk of the IBM 7040 associated with that console.

2.8 Use of MULTILIST

The MULTILIST System, a fundamental part of the MSPSF, is used by the programs of the Interactive Graph Theory System directly, so that a novice user does not need to know about the operations of MULTILIST. Users who are developing new graph-theoretic algorithms, writing new interactive ALLA programs, or writing new interactive user programs can benefit from the functions which are available from MULTILIST using the text console mode of the DEC-338 terminal.

The MULTILIST data file is the repository for source card images of any language, binary object decks, data items (such as graphs), operating system macros, and useful MULTILIST worker programs. All items contained in the file are referenced by descriptors or key words. Users may call upon standard worker programs for manipulating the descriptors assigned to various items, old items may be deleted, and new ones may be entered. There are programs which can be used in order to determine the contents of the file. Interested readers and users are referred to other documents of the MULTILIST Project.[16,24,45,46,72] Additional information specific to this system can be found in Appendices 2, 3, and 5.

2.9 Summary of Capabilities

This section serves as a review of this chapter by summarizing the essential features of the Interactive Graph Theory System as it presently operates. There is first a description from a user's point of view of what the system has to offer. Then, a systems designer's view is given indicating the existing subsystems.

2.9.1 What It Does

The Interactive Graph Theory System is used, augmented, and modified from the remote computer graphics terminal which includes display

screen, Teletype keyboard and printer, light pen, and pushbuttons. In the graphic mode, a complete graph drawing and editing facility is available. Graphs may be saved with associated descriptors and later restoration by description of any number of graphs may occur. A library of graph-theoretic algorithms is maintained so users may apply certain interactive algorithms to arbitrary graphs. Other algorithms aid in construction or recognition of particular types of graphs.

Complete programming facilities are available for the users who wish to develop new algorithms. File maintenance and examination functions may also be used. Knowledgeable programmers can alter and add to the basic system at various levels of implementation. Experimental systems programmers may use the system as a framework for studies in memory structures.

2.9.2 How It Is Done

The software of the Interactive Graph Theory System has all been generated on the IBM 7040 using FORTRAN IV, interactive ALLA, L^6, MULTILANG (MULTILIST Language), MAP, and PDPMAP. These programs run on the IBM 7040, intermediary PDP-8, and the DEC-338 graphics terminal. The operating system environment is a version of IBSYS modified for remote console use with fast-batch operation and integrated with the MULTILIST storage and retrieval system, forming the Moore School Problem Solving Facility. The program operating in the PDP-8 is dedicated to servicing consoles and interpreting a console control language. The DEC-338 can be used as an alphanumeric display console in order to use the MULTILIST system or any part of the operating system which is normally available to users of the IBM 7040. Of interest to users and programmers of the Interactive Graph Theory System is the availability of the compilers

and assemblers of those languages used in the system. There are some operating system macros (equivalent to a sophisticated job control language) which aid in these operations.

The other mode for the use of the DEC-338 is as a Graph Theory terminal where the small computer is under the control of a special-purpose executive program called DOGGIE (for Display of Graphs Graphical Interpretive Executive). As the name indicates, the program includes an interpreter of a specialized command language which controls the existence and display of graphs. Interactive programs may operate completely within the DEC-338 in which case they are composed of a mixture of PDP-8 machine language and DOGGIE interpreter language. A Graph Monitor with many facilities is provided as a basic system at the DEC-338. There is a capability for adding new user programs which have been assembled by PDPMAP on the IBM 7040 to the local system at the terminal.

Through the Graph Monitor a user can save a graph, restore any number of graphs, and initiate interactive execution of IBM 7040 programs. Each of these operations requires the running of a job on the IBM 7040, and this is set up by DEC-338 programs which prepare job input as if a user had typed his request. For interactive execution, the terminal is initially responsive only to commands from the IBM 7040. The program operating in the IBM 7040 at this point would be written in the interactive ALLA language. FORTRAN IV has been extended with data structure and associative processing components to form the language ALLA. This compiler-level language was then enriched with the primitives for interaction with the remote computer graphics terminal. Interactive ALLA programs may include output statements written in symbolic form for the DOGGIE interpreter in the DEC-338. There are interactive ALLA state-

ments which may be used to temporarily yield either partial or complete control to the terminal. A program in the IBM 7040 may call upon a DEC-338 user program as a subroutine. There are methods for the program in the IBM 7040 to receive input from programs and user actions in the DEC-338.

A collection of interactive ALLA programs has been written. Users who are programmers may add to this library from the terminal. The compiler of interactive ALLA programs consists of a preprocessing phase followed by application of the FORTRAN IV compiler. The preprocessor, written in L^6, transforms interactive ALLA statements which are not FORTRAN into FORTRAN SUBROUTINE and FUNCTION calls.

The underlying routines which implement the ALLA data structure are written in L^6. The memory structure which they employ can be varied without alteration of the semantics of the ALLA language. The underlying routines which implement the interactive statements are programmed in MAP assembly language, and they communicate with the DEC-338 via an associated input file and output file.

CHAPTER 3

THE ALLA LANGUAGE

3.1 Introduction

ALLA is an extension of the FORTRAN IV Language for the IBM 7040 which incorporates an additional data type, called "ENTITY", in order to handle data structures not representable in standard FORTRAN. ALLA includes all statements of the FORTRAN IV Language, plus a number of statements used in reference to entities.

ALLA does not explicitly refer to an entity, that is, there is no entity constant in the language (in the way that FORTRAN has integer constants, real constants, etc.) except for "UNDEF", which represents an undefined entity. An entity may be referred to by an entity variable or by a relation or association with an entity.

There are three types of entities: "ATOM", "PAIR", and "SET". An entity variable may, at any one time, name a particular atom, pair, or set, or it may name nothing (be undefined), i.e., it may have a value of UNDEF.

Each atom, pair, or set may have any amount of associated data. An associated datum is called a "PROPERTY", and is referenced by a PROPERTY NAME. The value of the property of an entity may be an integer, real, or logical constant, or it may be an entity. The type for each property name (one of the above four) must be declared in the same way that variable names are for consistent use throughout the same subprogram.

An entity of the type atom is one which has no structure other than its associated properties. A pair is a type of entity which, in addition to any properties, has a LEFT-ELEMENT and a RIGHT-ELEMENT, each of which may be an entity or be undefined. In Algebra, the term "ordered pair" is used to describe this structure.

A set, besides having associated properties, is a structure with any number of elements, each of which must be an entity. A set may have no elements, in which case it is called EMPTY. Although the word "set" is used, the implementation imposes an ordering to the elements, and so one may make use of the "list" nature of this structure. Also, membership in a set is not limited to a particular element appearing once. There are no restrictions on the structuring of data in this system. For example, a particular set may even be a member of itself three times. More important, however, is the unlimited hierarchy of the relationships which can be modelled in this structure.

This data structure can, of course, be easily used to represent a graph. We will usually define a graph according to Berge [4]: as an ordered pair, where the left-element of the pair is the set of vertices, and the right-element is the set of arcs. Each arc is an ordered pair, where the left element is the "from-vertex" of the arc, and the right element is the "to-vertex." Each vertex will ordinarily be an atom, but the ALLA data structure permits any type of entity as the member of a pair or set. Thus one could even represent a graph of graphs, or other interesting structures. More commonly, one finds the following structures appearing in various Graph Theory manipulations:

1. A set of arcs (where order is important) for a path.
2. A set of arcs as a property of a vertex - those outgoing arcs.
3. A set of vertices as a property of a vertex - those neighbors.
4. An integer as a property of a vertex - depth in a tree.

The remainder of this chapter describes those additions to FORTRAN IV which define ALLA. The reader should note the terminology is consistent with and in some places directly taken from the FORTRAN manual. The chapter ends with some programming examples which relate to Graph Theory.

3.2 Entity Variables

An entity variable name consists of one to six alphanumeric characters, the first of which is alphabetic. It may be subscripted, but one should note that some of the ALLA statements restrict the use of entity variables to those nonsubscripted ones. Nevertheless, it is therefore possible to represent 1-, 2-, or 3-dimensional arrays of entities. A 1-dimensional array of entities is somewhat like a set, except that an array size is fixed at compile-time, but there is no restriction on the number of elements in a set.

The programmer must declare those variable names he uses as entity variables in a "Type Declaration Statement" of the form:

ENTITY a, b, c,...

where a, b, c,... are entity variable names (or entity function names) appearing within the program.

Each variable name may optionally be subscripted with integer constants to specify dimensions.

Note that an ENTITY type declaration is in effect throughout the program, and may not be changed. The same entity variable may, however,

be used to name atoms, pairs, or sets.

Examples:

```
ENTITY S1, S2
ENTITY L(10), X, INARC, WIRE(3,5)
```

3.3 Entity Functions

The programmer may write FUNCTION subprograms of type "entity" in the same manner that he writes integer functions, real functions, etc. An entity function name consists of one to six alphanumeric characters, the first of which is alphabetic. When he refers to an entity function in a program, he must declare its name as an entity by using the ENTITY type declaration described in the preceding section.

The following form must be used as the first statement of an entity FUNCTION subprogram:

ENTITY FUNCTION name($a_1,a_2,\ldots,a_n$)

where name is the symbolic name of the single-valued function; and

$\underline{a_1},\underline{a_2},\ldots,\underline{a_n}$ are arguments, of which there must be at least one, which are nonsubscripted variable names, names of SUBROUTINE subprograms, names of FORTRAN functions, or names of library functions.

Examples:

```
ENTITY FUNCTION CROSS(S1,S2)
ENTITY FUNCTION SHPTHW(GRAPH,FROMV,TOV)
```

3.4 Entity Expressions

Basic FORTRAN IV includes two kinds of expressions: arithmetic and logical. A third kind of expression is included in ALLA: the entity expression. An entity expression may be an entity variable (possibly subscripted), an entity function, or an entity property

(defined in the next section). The entity constant representing a null or undefined entity - UNDEF - is also a valid entity expression. An entity function or property may include one or more entity expressions as arguments to any depth of call. This allows for such entity expressions as follows.

Examples:

```
X
INARC(V)
INARC(LELM(FIRSTA(V)))
LELM(LELM(LELM(LELM(X))))
GROUP(5,3,LELM(X))
```

where X and V are entity variables; INARC and FIRSTA are entity properties; and LELM and GROUP are entity functions.

The value of an entity expression is either an atom, a pair, a set, or UNDEF, for undefined.

3.5 Properties

A property name consists of one to six alphanumeric characters, the first of which is alphabetic. A property name must be declared as one of the four following types by a type declaration statement: INTEGER, REAL, LOGICAL, ENTITY. When referring to the property of an entity, the programmer uses the following property form:

name(expr)

where name is the property name; and

expr is any entity expression.

3.5.1 Property Assignment

The property form may be used to assign a value to a property.

name(expr) = val

where name and expr are as above; and

val is an expression whose type must match the type of property name name.

When a statement of this form is used in a program, the programmer must declare the property name as such by a type declaration of the form:

PROPERTY a,b,c,...

where a,b,c,... are property names (nonsubscripted) appearing within the program.

Examples:

PROPERTY RADIUS

PROPERTY LENGTH,WIDTH,HEIGHT,AREA

3.5.2 Property Use

The second way in which a property form may be used is within an arithmetic, logical, or entity expression. In this context, a property form is syntactically the same as a function call. This has been purposely done so that a programmer may write a statement such as:

AREA(RECT) = LENGTH(RECT) * HEIGHT(RECT)

where RECT is an entity variable, AREA is a real property, and LENGTH and HEIGHT are real.

The above statement is meaningful independently of whether LENGTH and HEIGHT are properties or functions. The programmer may decide at a later time which of the two possibilities is more appropriate.

If a property is used in an expression before its value has been defined, a default value is used as: zero for arithmetic properties, .FALSE. for logical properties, and undefined for entity properties. Such a default value is also used if a property is used after it has been removed from an entity (as described in the next section).

3.5.3 Property Removal

The programmer may decide that a particular property associated with a particular element is no longer wanted, and that property be removed. The following statement form is used to remove a property:

REMPROP p FROM expr

where p is a property name; and

expr is an entity expression.

If the value of expr is undefined, or if the entity named by expr does not have property p, the statement has no effect.

When a property is removed from an entity, only the association is lost. Removal of an entity property does not cause the deletion of the entity referenced. If the referenced entity were not referenced elsewhere in the ALLA structure then it would be left in a suspended useless state. This concept is further discussed in Sect. 3.7.

Examples:

REMPROP LENGTH FROM RECT

REMPROP NEIGH FROM NEIGH(V)

where RECT and V are entity variables, LENGTH is a property, and NEIGH is an entity property.

3.6 Entities and Properties - Some Examples

Although the programmer who is comfortable with list-processing or associative languages may find the concepts of entity and property rather natural, the FORTRAN-only programmer may be somewhat confused at this point in the text. Therefore some short examples are presented here to give the non-sophisticated reader enough confidence to continue further into this chapter.

The FORTRAN programmer should be familiar with the following form of arithmetic statement:

```
I = 5
```

The semantics of this statement gives the integer variable I the value of the integer constant 5. If the following arithmetic statement were subsequently encountered:

```
J = I
```

then the integer variable J would be given the value of the integer variable I, which is, of course, the integer constant 5. Now, there are two copies of the constant 5 which are values of I and J.

The above semantics are used throughout FORTRAN IV, but such a modus operandi is not preserved with the addition of the entity. First of all, there is no entity constant in ALLA comparable to "5" in normal FORTRAN IV. What must be understood is that an entity is not a number or numerical or even logical quantity; instead it is an abstract structure. It is therefore meaningless to write any of the following ALLA statements:

```
E1 = 5
E1 = E1 + E2
X = SQRT(E1)
```

where E1 and E2 are entity variables, and X is a real variable.

A numerical quantity may be associated with an entity by using an integer or real property of an entity. For example, if IP is an integer property name and RP is a real property name, then the above meaningless statements could be meaningfully rewritten as:

IP(E1) = 5 The integer property IP of the entity named by entity variable E1 is given the value 5.

IP(E1) = IP(E1) + IP(E2) The integer property IP of the entity named by entity variable E1 is updated by the value of the integer property IP of the entity named by the entity variable E2.

X= SQRT(RP(E1)) The real variable X is given the value of the square root of the value of the real property RP of the entity named by the entity variable E1.

The following ALLA statement is also meaningful in a sense different than in FORTRAN IV:

E1 = E2

where E1 and E2 are entity variables. Whereas a copy would be made if E1 and E2 were integer, real, or logical variables, when they are entity variables, the above statement causes the entity variable E1 to reference (or name, or point to) the entity referenced by E2. This interpretation is extended to include statements of the form:

E1 = expr

where E1 is an entity variable and <u>expr</u> is an entity expression.

Whenever an entity is used in ALLA, it is referenced through an entity variable, entity function, or entity property. In any of these cases, the atom, pair, or set being referenced is an abstract structure and is only being named or "pointed to" by the variable, function, or property.

As an example, suppose that entity variables A, B, and C each name a unique atom, and that NEXT1 and NEXT2 are entity properties, then the following ALLA statements set up an interesting relationship:

```
NEXT1(A) = B
NEXT1(B) = C
NEXT1(C) = A
NEXT2(A) = NEXT1(NEXT1(A))
NEXT2(B) = NEXT1(NEXT1(B))
NEXT2(C) = NEXT1(NEXT1(C))
```

The reader should note the last three statements could have been written alternatively as:

```
NEXT2(A) = C
NEXT2(B) = A
NEXT2(C) = B
```

On the other hand, the same effect could have been accomplished by defining NEXT2 as an entity function instead of using it as an entity property. The following is a complete ALLA entity function subprogram:

```
ENTITY FUNCTION NEXT2(X)
ENTITY X,NEXT1
NEXT2 = NEXT1(NEXT1(X))
RETURN
END
```

Note that in the definition of NEXT2 above, it is not necessary to distinguish between NEXT1 as an entity function or entity property.

The advantage of associating data by means of a property rather than a function is the savings in computation time. The advantage of using a function is the evaluation is performed for each instance of the name.

3.7 Entity Creation

3.7.1 Explicit Entity Creation

The following ALLA statements may be used to create explicitly a new entity:

```
CREATE ATOM ent
CREATE PAIR ent
CREATE SET ent
```

where ent is a nonsubscripted entity variable.

The first form above causes the entity variable ent to point to (or name) a new atom. The second form causes ent to point to a new pair, with undefined left-element and right-element. The third form causes ent to point to a new set, with no members (i.e., empty). In all three cases the new entity has no properties.

If the entity variable used in the CREATE statement had been pointing to some other entity, that old reference is lost. It is the duty of the programmer to delete unwanted entities before losing pointers to them. For example, consider the following sequence of statements:

```
CREATE ATOM X
CREATE SET X
```

After execution of the first statement entity variable X points to a new atom, and it is the only "handle" (umbilical cord, perhaps) at this time. Execution of the second statement creates a new set which is then pointed to by entity variable X, and that new atom is left "suspended in space" with nothing pointing to it. It is lost forever, but still occupies two words of storage space somewhere in the IBM 7040. The programmer has the power to express such a fiendish operation, but is advised against excessive abuse.

3.7.2 Implicit Entity Creation

The CREATE statement provides the programmer with sufficient power to create any basic entity, but since creation is a common operation there is an implicit method of creation. ALLA includes the following three built-in entity functions:

```
CRATOM(ent)
CRPAIR(ent)
CRSET(ent)
```

where ent is a nonsubscripted entity variable.
The first form above causes the entity variable ent to point to a new atom; the value of the function also points to the new atom. Similarly, the second and third forms create a new pair and a new set.

As in explicit creation, if ent had been the only handle on an entity prior to the creation operation, that entity is left in an unusable state.

Example:

```
RADIUS(CRATOM(C)) = 4.5
```

may be used as a short form of the following two statements:

```
CREATE ATOM C
RADIUS(C) = 4.5
```

where RADIUS is a real property name and C is an entity variable.
Note that in both of the alternative forms the entity variable C points to the created atom.

3.8 An Atom

An atom is the first type of entity available in an ALLA structure. The atom has no substructure other than any number of associated properties.

3.9 A Pair and Its Elements

A pair is the second type of entity available in an ALLA structure. The substructure of a pair consists of its left-element and right-element, each of which may be an ENTITY or be undefined. If one wishes to define an ordered pair of integers, he must associate an integer property with each of the two entity elements of a pair; the left- and right-elements may not themselves be integers. A pair may also have any number of associated properties.

The programmer refers to the left- and right-elements of a pair by the pair-element forms:

```
LELM(expr)
RELM(expr)
```

where _expr_ is any entity expression which names a pair.
If _expr_ is an entity expression which does not name a pair, both LELM(expr) and RELM(expr) are undefined.

The pair-element form may be used to assign a value to one of the elements of a pair:

```
LELM(expr) = ent
RELM(expr) = ent
```

where _expr_ and _ent_ are entity expressions.
If the term on the left side of the "=" is undefined, the statement has no effect.

Examples:

```
LELM(FIRSTA(V)) = FIRSTV(V)
RELM(CRPAIR(G)) = CRSET(ARCS)
```

where V, G, ARCS are entity variables, and FIRSTA and FIRSTV are entity properties.

The second way in which a pair-element form may be used is within an entity expression. In this context LELM and RELM may be considered built-in entity functions.

Examples:

VLIST(5) = LELM(G)

FIRSTV(V) = LELM(FIRSTA(V))

LELM(LELM(X)) = LELM(RELM(X))

where VLIST is a subscripted entity variable; V, G, X are entity variables; and FIRSTV and FIRSTA are entity property names.

3.10 A Set and Its Members

A set is the third type of entity available in an ALLA structure. The substructure of a set consists of any number of members, each of which must be an entity. A set may even have no members, in which case it is called "empty." A set may also have any number of associated properties.

Although a set normally denotes an unordered collection of members, the computer implementation being used for ALLA forces an ordering, and therefore one may use a set as a list - the order of the list is preserved throughout all references to the set.

Another extension to the classical notion of a set is the power in ALLA to allow an entity to be a member of a set any number of times. In fact the ALLA data structure has no limitation on the structuring of data. For example, a set may even be a member of itself three times; or the left-element of a pair may be the pair itself. More important than such strange relationships, however, is the unlimited hierarchy of relationships which can be modelled in an ALLA structure.

The remaining subsections of Section 3.10 plus Section 3.11 discuss the ways in which a set may gain or lose members. The explanation of how to reference the individual entities which are members of a particular set will be covered in a later section on THROUGH loops.

3.10.1 Insert a Member

Section 3.7 described how a new set may be created. When this is done the set is, of course, empty. The following statement form is used to insert an entity into a set as a new member:

```
INSERT expr1 INTO expr2
```

where expr1 and expr2 are entity expressions.

If the value of expr1 is undefined or if expr2 does not name a set, the statement has no effect. The entity which is inserted is made the "last" member of the set.

Examples:

```
INSERT V INTO NEIGH(RELM(A))
INSERT CRATOM(V) INTO CRSET(VS)
```

where V, A, VS are entity variables, and NEIGH is an entity property name.

3.10.2 Remove a Member

The following statement form is used to remove an entity from the membership in a set:

```
REMOVE expr1 FROM expr2
```

where expr1 and expr2 are entity expressions.

If the value of expr1 is undefined or if expr2 does not name a set, or if the entity named by expr1 is not a member of the set named by expr2, the statement has no effect. If the entity removed is a member of the

set more than once, only its first occurrence as a member is removed.

Examples:

REMOVE V1 FROM VERTS

REMOVE LELM(A1) FROM LELM(G)

REMOVE X FROM X

where V1, VERTS, A1, X are entity variables.

3.11 A Set as a Pushdown

A set may be referenced in ALLA as a pushdown list. The "push" operation may be used to insert an entity into a set as the first member of the set instead of being made the last member by the INSERT statement . The "pop" operation is an entity function whose value is the first member of the set, and application of the function also causes that first member to be removed from the set.

3.11.1 Push

The following statement form is used to push an entity onto a pushdown list (or set):

PUSH expr1 ONTO expr2

where expr1 and expr2 are entity expressions.

If the value of expr1 is undefined or if expr2 does not name a set, the statement has no effect. The entity which is inserted is made the "first" member of the set.

Examples:

PUSH NEIGH(V) ONTO PDL(I)

PUSH V ONTO CRSET(LIST)

where V and LIST are entity variables, PDL is a subscripted entity variable, I is an integer variable, and NEIGH is an entity property name.

3.11.2 Pop

The following expression form may be used within an entity expression to pop an entity from a pushdown list (or set):

POP(expr)

where expr is an entity expression.

If expr does not name a set or if the set named by expr is empty, the value of the above expression is undefined. The above function call removes the "popped" entity from the set as the set's first member.

Examples:

X = POP(LIST)

PUSH RELM(POP(ARCLST)) ONTO VLIST

where X, LIST, ARCLST, VLIST are entity variables.

3.12 Entity Deletion

The following statement form is used to delete an entity from the ALLA data structure:

DELETE expr

where expr is an entity expression.

If the value of expr is undefined, the statement has no effect.

When an entity is deleted all of its associated properties are also removed. In addition, if the entity had been the value of a property of some entity in the data structure, that value is made undefined. All those left-elements of pairs in the data structure which had the deleted entity as their value are made undefined. All those right-elements of pairs in the data structure which had the deleted entity as their value are made undefined. The deleted entity is also removed from all sets which contained the entity as a member in all such occurrences.

When an entity is deleted, although all references to it are removed from the ALLA structure, there may be any number of ALLA entity variables which were pointing to it. It is the programmer's obligation to avoid using such variables except to redefine their values. There are several exceptions to this warning which are discussed in Section 3.15.

Note that when a pair or set is deleted its elements or members are not automatically deleted. It would be rather unfortunate if this were not the case; for example, it would be impossible to delete an arc in a graph without deleting the two vertices which define it. In order to delete an entity and its substructure, a program must be used. This type of operation is not a primitive in ALLA since deletion may be done at various depths, and the programmer will likely decide for himself what is appropriate.

Examples:

```
DELETE A
DELETE LELM(FIRSTA(V))
DELETE CRATOM(A)
```

where A, V are entity variables, and FIRSTA is an entity property. Note the third statement has no practical value.

3.13 Entity Equality

The relational operators .EQ. and .NE. may be used within a logical expression to compare entity expressions. Since entities are not numeric quantities, the other relational operators may not be used to compare two entities (.GT., .LT., etc.). The following forms may be used:

(expr1 .EQ. expr2)

(expr1 .NE. expr2)

where expr1 and expr2 are entity expressions.

The first form is .TRUE. if and only if the value of expr1 is the same entity as the value of expr2. The second form is simply the negation of the first form.

3.14 Predicates

ALLA includes six built-in predicates or logical functions which are used to test entities. Examples of the use of these predicates follows the three subsections of this section.

3.14.1 NULL

The following form is a predicate:

NULL(expr)

where expr is an entity expression.

The value of the predicate is .TRUE. if and only if the value of expr is UNDEF. This predicate is not a necessary constituent in ALLA, since the following logical expression is equivalent:

(expr .EQ. UNDEF)

3.14.2 ATOM, PAIR, SET

The following forms are predicates:

ATOM(expr)

PAIR(expr)

SET(expr)

where expr is an entity expression.

The value of the first predicate is .TRUE. if and only if the value of expr is an atom. Similarly, the second predicate determines whether expr names a pair, and the third one determines whether expr names a set.

3.14.3 EMPTY

The following form is a predicate used for sets:

EMPTY(expr)

where expr is an entity expression.

The value of the predicate is .TRUE. if and only if either expr is undefined or the value of expr is a set which is empty.

3.14.4 MEMBER

The following form is a predicate used for sets:

MEMBER(expr1,expr2)

where expr1 and expr2 are entity expressions.

The value of the predicate is .TRUE. if and only if the value of expr2 is a set and the value of expr1 is an entity which is a member of that set.

Examples:

```
IF (NULL(RELM(A))) GOTO 10
K = .NOT.PAIR(A)
IF (SET(B) .AND. MEMBER(A,B)) I=1
```

where A, B are entity variables, K is a logical variable, and I is an integer variable.

3.15 THROUGH Statement

The THROUGH statement is used for control in an ALLA program in order to refer to the members of a set. It is an extension of the DO statement of FORTRAN IV in that it is used to cause repetitive execution of a series of statements. The following is a form of this statement:

```
THROUGH n FORALL ent IN expr
```

where n is a statement number;

ent is a nonsubscripted entity variable; and

expr is an entity expression.

This statement results in the execution of the statements that follow the THROUGH, in the range, up to and including the statement numbered n. This iteration occurs once for each member of the set which is the value of expr. If the value of expr is not a set, or if it is an empty set, the statements within the range are not executed. Otherwise, the bound entity variable ent takes on the value of the first member of the set on the first iteration. Then after control continues past statement n, the statements within the range are executed, with the value of ent as the entity which is the second member of the set. This process continues until control passes statement n with the iteration having been performed on the last member of the set; in this case, control then passes to the statement following statement n.

The bound variable of a THROUGH is the entity variable ent. Throughout the range of the THROUGH, it may be used in any entity expression. If control leaves the range of the THROUGH in any way, the variable may then be considered free, and it may be used in further entity expressions. If the iteration occurs for all members of a set, and if the value has not been altered, then this variable will attain the value of the last member of the set. If no iteration occurs, the value of the variable is undefined.

The programmer is permitted to alter the value of the bound variable within the range of a THROUGH statement since the variable is not used for the purposes of determining the "next" member.

The programmer may alter the membership of the set being used for a THROUGH statement, even within the range of the THROUGH. The selection of the "next" element of a set occurs as a dynamic computation for each iteration around the loop. This generality even permits the deletion of the set being used within the range.

THROUGH statements may be nested among themselves or with DO statements. However, control may not be transferred into the range of a THROUGH from outside the range.

The same rules apply for the statement that terminates the range of a THROUGH as with the range of a DO, i.e. such a statement must be an executable statement, and cannot be an arithmetic-IF nor a GOTO.

Examples:

```
THROUGH 10 FORALL A IN ARCS
THROUGH 250 FORALL V1 IN LELM(G)
```

where A, ARCS, V1, G are entity variables.

3.16 Property Handling

Section 3.5 described how entities may have an arbitrary amount of associated data, known as properties. It indicated how the association is made, how it is removed, and how the programmer references or uses properties. These operations are not powerful enough to express all important manipulations on properties. The additional necessary feature which the programmer needs is the ability to handle those properties which are associated with an entity, without "knowing" anything about how many there are, which ones are used, or what their values are. This type of information is needed to output an entire structure without the knowledge of all properties being used. A simpler example where such a feature is needed is in a routine to copy an entity including

all of its properties.

This section describes four special built-in functions which give the programmer the power to handle those existing properties of an entity. A special subroutine is also described which can be used for setting the value of an arbitrary property of an entity. An illustrative example is given at the end of the section.

3.16.1 Property Set

Properties are associated with an entity within the ALLA data structure in the form of a set, much in the same way that a set of entities is modeled. This makes it possible to use the THROUGH statement to sequence through all the properties (property elements) in the "property set" of an entity. The following special built-in function is used in a THROUGH statement to refer to the property set of an entity:

```
THROUGH n FORALL prel IN PRSET(expr)
```

where n is a statement number;

prel is a nonsubscripted entity variable; and

expr is an entity expression.

The above THROUGH statement is meaningful only when the value of expr is an atom, pair, or set. Otherwise, the statements within the range of the THROUGH are not executed. On each iteration the bound variable prel refers to a "property element." A property element is not an ALLA entity, and so it may not be manipulated in any standard ALLA statement. Instead there are two special built-in functions which may be applied to property elements. These special functions may only be used in this way.

Each property element represents one existing property of an entity.

3.16.2 Property Name

The following special built-in function is used to determine the name of the property of the association represented by a property element:

PRNAME(prel)

where prel is an entity variable whose value is a property element. The value of this function is not a BCD representation, but a quantity which can substitute for a property name in a function call, in a subroutine call, in a REMPROP statement, or in the special built-in functions PRENT or PRBCD (introduced below).

This function is used in the copying of an entity.

3.16.3 Property Value

The following special built-in function is used to determine the value of the property of the association represented by a property element:

PRVAL(prel)

where prel is an entity variable whose value is a property element. The value of this function is a direct (36-bit) copy of the value of the property, and it is of type integer or entity. Care must be taken by using EQUIVALENCE statements when properties are of types logical or real, and their values are to be used within expressions.

This function is used in the copying of an entity.

3.16.4 Setting the Value of a Property

Section 3.5.1 introduced the form used to assign a value to a particular property which has been declared as such. An alternate form

is available for the same purpose when the property name is unknown at the time of compilation. The following subroutine call may be included in an ALLA program:

```
CALL SETVAL(name,expr,val)
```

where name is either a property name, the PRNAME function of a property element, or an expression whose value is the BCD representation of a property name (left adjusted and padded with blanks).

expr is any entity expression; and

val is an expression whose type must match the type of property referenced by name.

This subroutine call sets the value of the property referenced by name of the entity which is the value of expr to the value of val.

A subroutine call of this type is used in the copying of an entity. It may also be used to create arbitrary properties of an entity at execution time.

3.16.5 Property Type

The following built-in logical function determines whether a particular property is of the type entity:

```
PRENT(name)
```

where name is either a property name, the PRNAME function of a property element, or an expression whose value is the BCD representation of a property name (left adjusted and padded with blanks).

The value of the PRENT function is .TRUE. if and only if the property name is used for an entity property.

This function is used in the recursive deletion of a structure.

3.16.6 BCD-Name of a Property

The following built-in integer function yields a quantity which is the 6-character BCD value of the name of the property:

PRBCD(name)

where name is either a property name, or the PRNAME function of a property element.

If the name consists of less than six characters, its code is left-adjusted and padded with blanks. If the given argument does not represent a property name, the value of the function is zero.

This function is used in order to output the properties of an entity.

3.16.7 Illustrative Example: COPY

The following is a listing of the ALLA program used to copy an entity. It is an entity function named COPY and is self-explanatory.

```
          ENTITY FUNCTION COPY(E)
          ENTITY E,M
          IF (ATOM(E)) GOTO 10
          IF (PAIR(E)) GOTO 20
          IF (SET(E)) GOTO 30
          COPY = UNDEF
          RETURN
   10     CREATE ATOM COPY
          GOTO 50
   20     CREATE PAIR COPY
          LELM(COPY) = LELM(E)
          RELM(COPY) = RELM(E)
          GOTO 50
   30     CREATE SET COPY
          THROUGH 40 FORALL M IN E
   40     INSERT M INTO COPY
   50     THROUGH 60 FORALL M IN PRSET(E)
   60     CALL SETVAL(PRNAME(M),COPY,PRVAL(M))
          RETURN
          END
```

3.17 Uses of an Entity

For efficient processing in certain types of problems, the ALLA programmer is given the primitives to determine what entities make reference to or use a particular entity. An entity may be referenced or used within the ALLA Data Structure in any of the following ways: as the left- or right-element of a pair, as a member of a set, and as the value of an entity property. Note that an ALLA entity variable pointing to an entity is not considered a use of that entity.

This section describes three special built-in functions which give the programmer the power to handle the uses of any existing entity. An illustrative example is given at the end of the section.

3.17.1 Use-Set

Each existing entity has an associated use set, which is modeled in the same format as a set of entities. This makes it possible to use the THROUGH statement in order to sequence through all the uses (use-elements) in the use-set of an entity. The following built-in function is used in a THROUGH statement to refer to the use-set of an entity:

```
THROUGH n FORALL usel IN USESET(expr)
```

where n is a statement number;

usel is a nonsubscripted entity variable; and

expr is an entity expression.

The above THROUGH statement is meaningful only when the value of expr is an atom, pair, or set. Otherwise, the statements within the range of the THROUGH are not executed. On each iteration the bound variable usel refers to a "use-element." A use-element is not an ALLA entity, and so it may not be manipulated in any standard ALLA statement. Instead,

there are three special built-in functions which may be applied to use-elements. These special functions may only be used in this way.

Each use-element represents one existing use of an entity. There is one use-element associated with each instance where an entity is a member of a particular set. If an entity is both a left- and right-element of a particular pair, its use-set contains two use-elements which represent the use of the entity by the pair. Likewise, there is one use-element associated with each instance where an entity is the value of an entity property of a particular entity.

3.17.2 Type of Use

The following built-in integer function determines the type of use a particular use-element represents:

USETYP(usel)

where usel is an entity variable whose value is a use-element. The value of this function may attain only one of four values:

USETYP(usel) = 1 when the value of usel is a use-element representing a use as the value of an entity property.

USETYP(usel) = 2 when the value of usel is a use-element representing a use as an element of a pair.

USETYP(usel) = 3 when the value of usel is a use-element representing a use as a member of a set.

USETYP(usel) = 0 when one of the above conditions is not met, i.e. in an error situation.

3.17.3 Entity Where Used

The following built-in entity function determines the host entity in the relationship represented by a particular use-element:

USEENT(usel)

where usel is an entity variable whose value is a use-element.

The value of this function is the entity where the use occurs. If the value of usel is a use-element representing a use as the value of an entity property, the value of USEENT(usel) is that entity with which the property is associated. If USEENT is applied to an argument which is not a use-element, its value is undefined. (This corresponds to the case when USETYP(usel) = 0.)

3.17.4 Property Where Used

The following built-in function is used to determine which property relationship is represented by a particular use-element:

USEPR(usel)

where usel is an entity variable whose value is a use-element representing a use as the value of an entity property.

The value of this function is a property element. Recall that a property element is not an ALLA entity; it may only be used in the special built-in functions PRNAME and PRVAL described in Sections 3.16.2 and 3.16.3. If USEPR is applied to an argument which is not a use-element representing a use as the value of a property, its value is undefined. (This corresponds to the cases when USETYP(usel) = 0,2,3.)

3.17.5 Illustrative Example

The following is a listing of an ALLA subroutine which demonstrates the handling of the uses of an entity. The subroutine named UNUSET has one argument which is an entity. The subroutine removes that entity from every set in which it is a member.

```
      SUBROUTINE UNUSET(E)
      ENTITY E,U
      THROUGH 10 FORALL U IN USESET(E)
      IF (USETYP(U) .NE. 3) GOTO 10
      REMOVE U FROM USEENT(U)
10    CONTINUE
      RETURN
      END
```

3.18 Miscellaneous Restrictions

This section indicates some restrictions on the syntax of ALLA programs which are not found in FORTRAN IV.

3.18.1 Reserved Words

The following is an alphabetical list of words which the ALLA programmer may not use as his own variable names or function names:

ATOM	MEMBER	REMOVE
CRATOM	NULL	REMPRO*
CREATE	PAIR	SET
CRPAIR	POP	SETVAL
CRSET	PRBCD	STLELM*
DELETE	PRENT	STRELM*
EMPTY	PRNAME	UNDEF
ENTITY	PRSET	USEENT
FORALL	PRVAL	USEPR
FORNXT*	PUSH	USESET
INSERT	RELM	USETYP
LELM		

* These words are not explicitly used in ALLA statements, but their use is restricted due to the underlying implementation of the ALLA compiler and execution-time system.

3.18.2 Statement Numbers

Statement numbers in an ALLA program may range between 1 and 99000, i.e. a statement number may not exceed 99000.

3.18.3 Logical-IF Statement

The executable statement which is to the right of the logical expression in a Logical-IF statement may not be any of the following ALLA statement forms:

CREATE ...

INSERT ...

PUSH ...

REMOVE ...

REMPROP ...

DELETE ...

LELM(expr) = ...

RELM(expr) = ...

name(expr) = ... where name is a property name

THROUGH ...

3.18.4 COMMON

Since blank COMMON is used for the maintenance of the ALLA data structure, the programmer may not use it. Labeled COMMON may be freely used.

3.18.5 Comments

As in FORTRAN IV, ALLA programs may include comment cards which are denoted by the letter "C" in column 1 of the source card image. In addition, the ALLA compiler considers a card image to be a comment if it begins with the special character "*" in column 1.

3.19 Programming Examples

This section includes ALLA subroutines and functions which have some practical value in Graph Theory. A graph will be assumed to have the structure given in Section 3.1.

3.19.1 Cardinality of a Set

The following integer function yields the number of members of the set given as the argument.

```
          INTEGER FUNCTION CARD(S)
          ENTITY S,E
          CARD = 0
          THROUGH 1 FORALL E IN S
   1      CARD = CARD + 1
          RETURN
          END
```

3.19.2 Incoming and Outgoing Arcs

The following subroutine accepts one argument which is assumed to be a directed graph. The routine defines two properties of each vertex of the graph. The INARC property is used to refer to the set of those arcs which enter into a vertex. The OUTARC property is used to refer to the set of those arcs which are defined to emanate from a vertex.

```
          SUBROUTINE INOUT(G)
          ENTITY G,X,V,A,INARC,OUTARC
          PROPERTY INARC,OUTARC
          THROUGH 10 FORALL V IN LELM(G)
          INARC(V)  = CRSET(X)
   10     OUTARC(V) = CRSET(X)
          THROUGH 20 FORALL A IN RELM(G)
          INSERT A INTO INARC(RELM(A))
   20     INSERT A INTO OUTARC(LELM(A))
          RETURN
          END
```

3.19.3 Make a Graph Undirected

Once the above subroutine has been applied to a graph, each vertex has a set of incoming arcs and outgoing arcs. The following subroutine may be subsequently applied to the same graph in order to make each incoming arc which is not a loop into an outgoing one, and remove the set of incoming arcs. Thus, according to the OUTARC property, the graph has been made undirected.

```
          SUBROUTINE UNDIR(G)
          ENTITY G,V,A,INARC,OUTARC
          PROPERTY INARC
          THROUGH 20 FORALL V IN LELM(G)
          THROUGH 10 FORALL A IN INARC(V)
          IF (LELM(A) .EQ. V) GOTO 10
          INSERT A INTO OUTARC(V)
   10     CONTINUE
          DELETE INARC(V)
   20     REMPROP INARC FROM V
          RETURN
          END
```

3.19.4 Connected Components of a Graph

After three rather simple examples, the following useful function plunges into significant computation and complexity. The function accepts one argument which is assumed to be a directed graph. Treating

```
          ENTITY FUNCTION CCNCOM(G)
          ENTITY G,V,X,VERTS,ARCS,A,TEMP,TEMPV,C,N
          ENTITY INARC,OUTARC,GROUP
          PROPERTY GROUP
*
          CALL INOUT(G)
          CALL UNDIR(G)
          VERTS = LELM(G)
          ARCS  = RELM(G)
          THROUGH 10 FORALL V IN VERTS
   10     GROUP(V) = UNDEF
          CREATE SET CONCOM
          THROUGH 80 FORALL V IN VERTS
          TEMP = GROUP(V)
          IF (TEMP .NE. UNDEF) GOTO 30
          LELM(CRPAIR(TEMP)) = CRSET(X)
          INSERT TEMP INTO CONCOM
          GROUP(V) = TEMP
          INSERT V INTO LELM(TEMP)
   30     THROUGH 70 FORALL A IN OUTARC(V)
          N = LELM(A)
          IF (N .EQ. V) N = RELM(A)
          IF (N .EQ. V) GOTO 70
          IF (GROUP(N) .NE. UNDEF) GOTO 60
          GROUP(N) = TEMP
          INSERT N INTO LELM(TEMP)
          GOTO 70
   60     IF (GROUP(N) .EQ. GROUP(V)) GOTO 70
          THROUGH 65 FORALL TEMPV IN LELM(TEMP)
          GROUP(TEMPV) = GROUP(N)
   65     INSERT TEMPV INTO LELM(GROUP(N))
          DELETE LELM(TEMP)
          DELETE TEMP
          TEMP = GROUP(N)
   70     CONTINUE
   80     CONTINUE
*
          THROUGH 100 FORALL C IN CONCOM
  100     RELM(C) = CRSET(X)
          THROUGH 110 FORALL A IN ARCS
  110     INSERT A INTO RELM(GROUP(LELM(A)))
          THROUGH 120 FORALL V IN VERTS
  120     REMPROP GROUP FROM V
          RETURN
          END
```

the graph as if it is undirected, the algorithm yields a set of graphs, with the vertices and arcs of the given graph, which constitutes the set of connected components of the graph.

The algorithm begins by establishing the OUTARC's of all vertices of the graph considered as undirected (see the two previous examples). Next, the variables VERTS and ARCS are assigned to refer to the set of vertices and the set of arcs. Each vertex is then defined to have an associated entity property named GROUP, which is initially undefined.

After creating the set which will be used as the result of the function, the major computation loop begins. In this loop, every vertex will be considered. The loop begins by determining whether that vertex has already an associated group. A group is a potential connected component. If it doesn't, a new potential graph is created and the vertex being considered is made to belong to the new graph.

At statement 30, an inner loop begins which scans all outgoing arcs of the vertex in consideration. This is being done in order to scan all first neighboring vertices of the one in consideration. Each neighbor is considered. If the neighbor in consideration does not yet have an associated group, it is made to be a member of the same group as the vertex currently being considered. If the neighbor in consideration already belongs to the same group as the vertex in consideration, all is well. If, however, the neighbor's group differs, then the two groups must be merged. The group of the neighbor remains and all vertices which are members of the group of the current vertex are moved into neighbor's group. The potential graph is deleted, and the next neighbor is then considered.

After all neighbors of all vertices have been scanned in the above manner, the connected components have been established. The arcs of each component are then defined by sorting each arc by its associated group. Finally, the algorithm ends by removing the GROUP property from all vertices, since it was only being used as an aid to the speed of the algorithm.

3.19.5 Undirected Tree

The following example is a logical function which determines whether its given argument considered as an undirected graph is more specifically a tree. The function makes use of two previously presented functions. First, if the given graph has more than one connected component it cannot be a tree. Second, in order to be a tree it can have no cycles, and this can be shown intuitively to be equivalent to having one less arc than the number of vertices.

```
      LOGICAL FUNCTION NDTREE(G)
      INTEGER CARD
      ENTITY CONCOM,G,C,G1
      NDTREE = .FALSE.
      C = CONCOM(G)
      IF (CARD(C) .NE. 1) GOTO 100
      IF (CARD(RELM(G)) .GE. CARD(LELM(G))) GOTO 100
      NDTREE = .TRUE.
100   THROUGH 110 FORALL G1 IN C
      DELETE LELM(G1)
      DELETE RELM(G1)
110   DELETE G1
      RETURN
      END
```

3.19.6 Directed Tree

This final example is included without explanation for the conscientious reader. It is a logical function of two actual arguments, but with the third argument of the function used to return parts of the answer. The first argument is assumed to be a directed graph where OUTARC's have been defined (by INOUT, for example). The second argument is assumed to be one of the vertices of the graph considered as the potential root. The algorithm determines whether the graph is a directed tree with the given root. A by-product of the algorithm returns the depth of the tree in the function's third argument. Also each vertex of the tree has an associated depth.

```
              LOGICAL FUNCTION DTREE(G,ROOT,D)
              ENTITY G,ROOT,CLEVEL,NLEVEL,V,NEXTA,OUTARC
              INTEGER DEPTH,D
              PROPERTY DEPTH
              DTREE = .FALSE.
              IF (.NOT.MEMBER(ROOT,LELM(G))) GOTO 80
              THROUGH 10 FORALL V IN LELM(G)
       10     DEPTH(V) = 0
              DEPTH(ROOT) = 1
              INSERT ROOT INTO CRSET(CLEVEL)
              D = 0
       20     IF (EMPTY(CLEVEL)) GOTO 40
              D = D + 1
              CREATE SET NLEVEL
              THROUGH 30 FORALL V IN CLEVEL
              THROUGH 30 FORALL NEXTA IN OUTARC(V)
              IF (DEPTH(RELM(NEXTA)) .NE. 0) GOTO 60
              DEPTH(RELM(NEXTA)) = D + 1
       30     INSERT RELM(NEXTA) INTO NLEVEL
              DELETE CLEVEL
              CLEVEL = NLEVEL
              GOTO 20
       *
       40     THROUGH 50 FORALL V IN LELM(G)
       50     IF (DEPTH(V) .EQ. 0) GOTO 70
              DTREE = .TRUE.
              GOTO 70
       60     DELETE NLEVEL
       70     DELETE CLEVEL
       80     RETURN
              END
```

CHAPTER 4

DISPLAY OF GRAPHS GRAPHICAL INTERPRETIVE EXECUTIVE

4.1 Introduction

The DEC-338 has two distinct roles in the Interactive Graph Theory System. It may serve as a text console for the purposes of writing new interactive ALLA programs, local user programs, interactive user programs, etc. It may be used to perform storage and retrieval of text files, editing, compiling, assembling, etc. The more fitting use of this graphical terminal is in its other role as a special-purpose machine responsible for the display of graphs. During this mode of operation an executive program called DOGGIE - for Display of Graphs Graphical Interpretive Executive - is in control of the DEC-338. A list of the important services performed by DOGGIE follows:

1. The heart of DOGGIE is the interpreter of the DOGGIE Language, which accepts sequences of 12-bit words which cause parts of graphs to be created, altered, deleted, and displayed in a variety of manners.
2. DOGGIE manages all input/output of the DEC-338 by handling interrupts from the various display flags, the 637 Dataphone Interface, the DF32 Minidisk, and the Teletype.
3. DOGGIE manages the display of the graph on the "paper" with four window sizes available for viewing all or any part of the paper.
4. Light pen tracking is performed with optional horizontal and/or vertical constraints on a pseudo-pen-point.
5. DOGGIE interprets light pen hits as a result of pointing at displayed parts of a graph.

6. DOGGIE helps in the management of the pushbuttons of the pushbutton box attached to the DEC-338.

7. DOGGIE handles overlays of program segments by name by interfacing with the PDP-8 Disk Monitor System. Programs may be a mixture of PDP-8 machine language and DOGGIE commands.

8. DOGGIE includes facilities for the read out of status information concerning the state of DOGGIE and its existing graph.

All of the above features will be explained during the remainder of the chapter.

For the purposes of an introductory view, the interpreter of DOGGIE should be considered as a special purpose machine dedicated to the display of graphs. Its basic or primitive operations are controlled by a machine language which is accepted as sequences of 12-bit words. Each command to DOGGIE consists of some number of these words which are bit patterns. For purposes of description and also actual use in programs, the DOGGIE Language is written symbolically. For example the two words 3440 (octal) and 0000 constitute a command which causes all arrows of (directed) arcs existing in the graph to blink. The symbolic DOGGIE command which is equivalent to and in fact represents these two words is:

START BLINK ARROW ARC, ALL

Since the bit patterns lack mnemonic value, and since the symbolic form is used by the programmer, the DOGGIE commands are presented completely symbolically. There is a simple method of translating symbolic DOGGIE commands into bit patterns, and this is given in a separate section.

4.2 Interaction Through Communication Cells

The DOGGIE Language is a machine-independent language for the control of the display of graphs. The language in the pure sense is not interactive - it is only an output language. It becomes interactive when used in conjunction with other languages which include control specification. The DOGGIE language is used in two different environments within the Interactive Graph Theory System: First, it is imbedded into the ALLA Language at the compiler language level in the IBM 7040. Second, it is imbedded into PDPMAP Assembly Language through macros for use in assembly language programs operating in the DEC-338. The use of the DOGGIE Language in either environment has the same meaning, which is to direct DOGGIE to perform commands which define, alter, and display graphs.

The writing of interactive programs is somewhat different in the two environments, but the basic idea is common to both languages. The input aspect of the interaction is accomplished by programming in the host language the observation of communication cells. These cells reflect the status of the DEC-338 and DOGGIE back to the programmer. The information contained in these cells includes:

1. Current Graph Display Status - intensity, window size and position.
2. Light pen tracking indicators.
3. Indication of the amount of available free blocks.
4. Light pen hit information.
5. Complete status output for a vertex or arc.

A communication cell is a rather natural concept for use in the DEC-338, for in that environment it is simply an accessible location on the initial page of field 0. User programs operating in the DEC-338 are written with references to the symbolic names of these cells. There are

some communication cells in the DEC-338 which are used as indirect addresses of subroutines which DOGGIE performs such as the subroutine to send an 8-bit character to the Dataphone. A number of communication cells of this type are not duplicated in the ALLA environment.

Communication cells in the ALLA environment are referenced as any other FORTRAN integer variable or logical variable. They are automatically declared in each ALLA subprogram, so the programmer simply references these cells by symbolic name.

Many of the communication cells in the 7040 are essentially copies of the "real" cells in the DEC-338. The "real" introduced here means real-time. For example, a pair of communication cells indicate the position of the pseudo-pen-point linked to the light pen tracking cursor. In the DEC-338, programs observing these cells may do so in real-time. However, the communication cells of the DEC-338 are copied over to the 7040 only when requested by certain statements in interactive ALLA programs. Since this copying operation takes about one or two seconds to complete, and since tracking may alter the position of the pseudo-pen-point every few milliseconds, the communication cells in the 7040 cannot reflect this data in real-time. There are a few communication cells in the 7040 not found in the DEC-338 since some common functions are directly performed by subroutine calls in the smaller machine, such as the checking for Teletype input.

4.3 Organization

All DOGGIE commands implicitly refer to a scratchpad "paper" on which a single graph may be defined. It is only one graph in the sense that there is no facility for hierarchical grouping, however, the single graph may consist of any number of disjoint components which may give

the effect of displaying more than one graph at a time. The graph may consist of any number of vertices or arcs, and it may be only partially defined at any time, since it is permissible to define an arc in terms of vertices which do not yet exist. There is separate control over which parts of the defined graph are to be displayed.

The graph maintained by DOGGIE is built, modified, and deleted through interpreted DOGGIE commands. The command language includes elements which affect the gross aspects of either the existing graph, the displayed graph, or the way in which the graph is being displayed. A group of commands may refer to a unique vertex or arc by internal name or to all existing vertices or arcs. The option is also available for DOGGIE to supply a created internal name when a vertex or arc is defined.

4.3.1 Paper, Window, and Screen

The drawing area where a graph may be defined is a two-dimensional square grid called the "paper". The x-coordinate and y-coordinate of the paper both range over the integers between 0 and 1023_{10}. Figure 4-1 shows the coordinates of the paper with a dot shown at position (y,x) = (512, 256).

The distance between two adjacent paper positions on the same coordinate axis is called a <u>paper unit</u>.

Graphs are always defined in terms of paper y and x coordinates, but the display of the graph as put on the display screen may vary according to the window size and position. The viewing screen on the DEC-338 is a 9 3/8" square and should be considered as a square window on the paper. There are four window sizes available for viewing (the size is the length of one of the sides of the window):

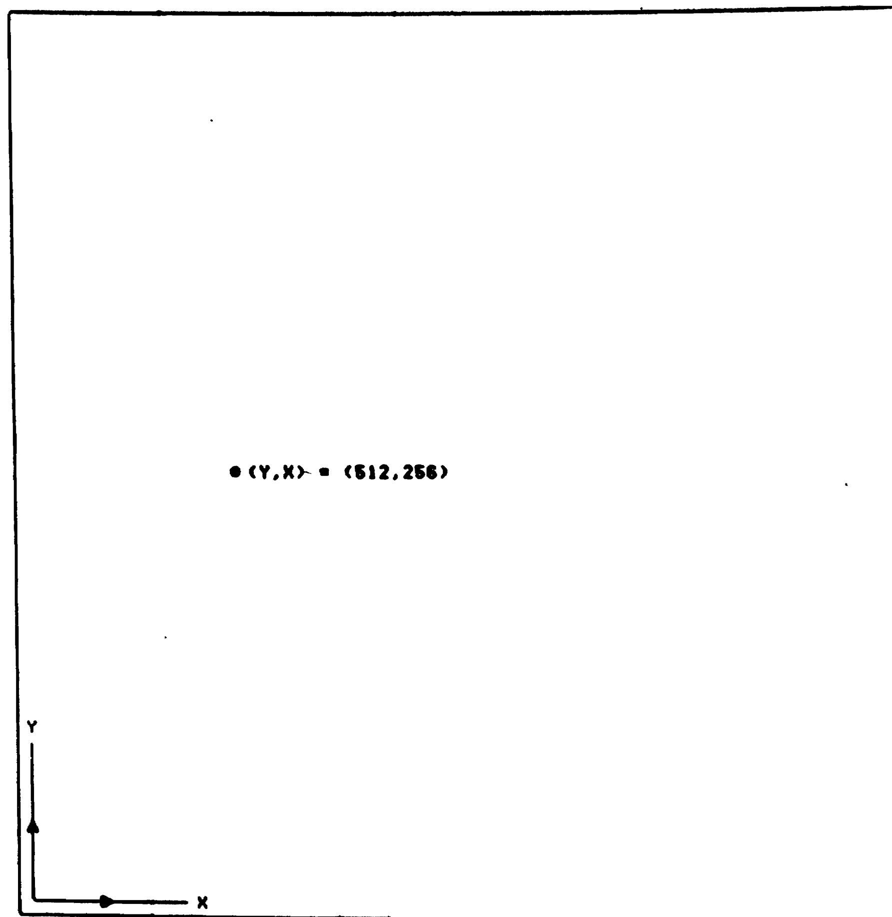


Figure 4-1 The Paper

1) full paper - 1024 paper units (largest window)

2) half paper - 512 paper units

3) one-fourth paper - 256 paper units

4) one-eighth paper - 128 paper units (smallest window)

When a full paper window is used, the paper and window are the same size, and the window cannot be moved at all. With any other window size, only part of the paper may be viewed, and the window's position on the paper may be varied as long as the window always remains entirely on the paper, never crossing an edge. The window's position is specified as a paper position which is the window's center (this is actually one-half of a paper unit in y and x greater than the true center since all windows have an even number of paper units in width).

A summary of the interesting relationships between the window and the paper is given in Table 4-1.

Window Size	Window Size in paper units	Minimum Window Position (y or x) paper coords.	Maximum Window Position (y or x) paper coords.	Actual Paper Size (inches)	Actual Window Size (inches)
FULL	1024	512	512	9 3/8	9 3/8
HALF	512	256	768	18 3/4	9 3/8
FOURTH	256	128	896	37 1/2	9 3/8
EIGHTH	128	64	960	75	9 3/8

Table 4-1 Relationships Between Window and Paper

There are DOGGIE commands to set the window size, to set the window position, and to move the window.

It is often useful to refer to a particular position on the physical screen of the display. For such pruposes, one should refer to the (y,x) screen coordinates. Consistent with paper coordinates, the screen coordinates range between 0 and 1023; paper coordinates and screen coordinates are coincident when a full-paper window is used.

When the window is some size other than full-paper, the screen coordinates form a more dense grid than paper coordinates (in units per inch). Therefore, when a vertex is positioned on the paper according to some screen coordinate, a round-off may be introduced.

4.3.2 Vertices

A vertex is defined to exist at some (y,x) position on the paper. Each vertex has a unique vertex internal name, which is some integer between 1 and 4095. Each vertex has one of eight possible shapes for display purposes. One of these shapes is null; but the other seven are distinguishable visible forms for displaying those graphs where vertices may be of various types, such as a Moore state graph. Figure 4-2 shows the seven visible shapes being used.

Each vertex may have an associated text label consisting of up to 27 characters. The label may be located anywhere on the paper relative to the position of the vertex. This relative position is called the "offset" of the label.

A vertex may exist without being displayed. Also, the text label of a vertex may exist without being displayed. Figure 4-3 depicts vertices as they appear along with arcs.

4.3.3 Arcs

An arc is defined to exist as a connecting line between two vertices. Each arc has a unique arc internal name, which is some integer between

SHAPE 1

SHAPE 2

SHAPE 3

SHAPE 4

SHAPE 5

SHAPE 6

SHAPE 7

Figure 4-2 Vertex Shapes

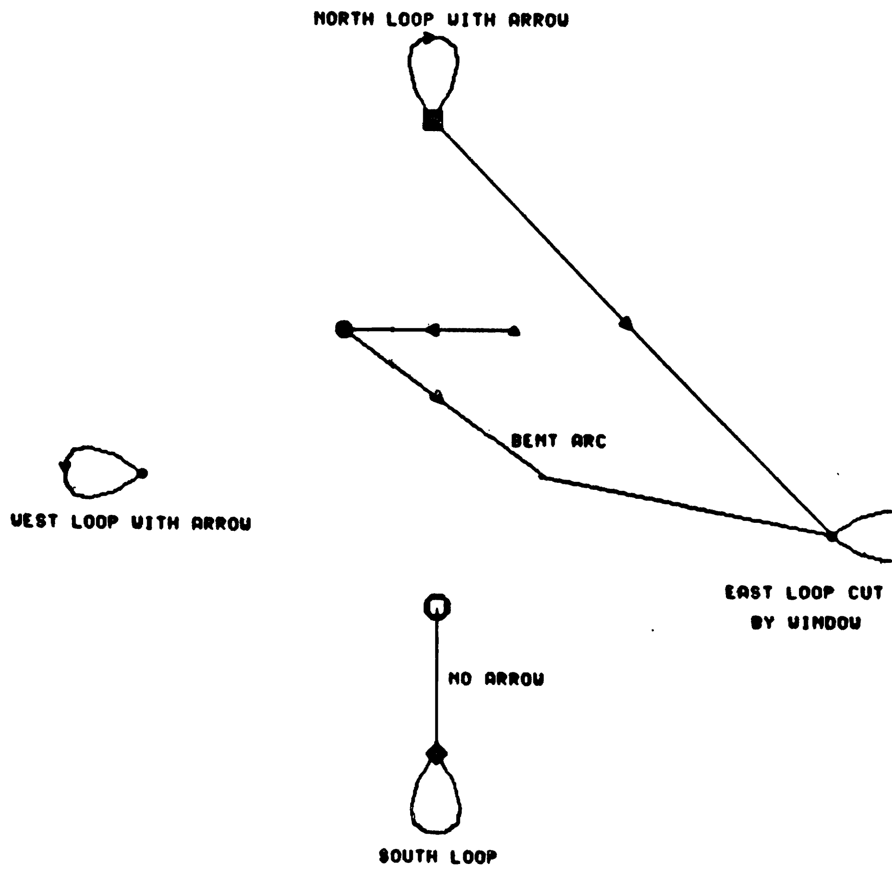


Figure 4-3 Display of Vertices and Arcs

1 and 4095. Each arc has a "from-vertex" which is a vertex internal name (between 1 and 4095), and a "to-vertex" which is also a vertex internal name. When the two vertex internal names are the same, the arc is a loop.

An arc may exist even when the vertices on which it is defined do not exist. The vertices must exist, however, for the arc to be displayed.

An arc which is not a loop is displayed as a straight line between its two vertices. A loop is displayed as a closed curve somewhat elliptical in appearance. In order to draw good looking graphs four loop orientations are available: East, North, West, South.

Each arc may have an arrow. When it does, and when the arc is displayed, the arrow is displayed at the midpoint of the arc pointing in the direction toward the "to-vertex." When an arrow is displayed it is drawn in one of eight orientations according to the angle of the arc.

Each arc may have an associated text label consisting of up to 21 characters. The label may be located anywhere on the paper relative to the position of the midpoint of the arc. This relative position is called the "offset" of the label.

An arc may exist without being displayed. Also, the text label of an arc may exist without being displayed. Figure 4-3 shows the display of the various types of vertices and arcs.

4.3.4 Created Internal Names

It is useful to have DOGGIE provide created internal names for vertices and arcs, particularly for an interactive user program where new entities must be created. DOGGIE maintains a next created vertex internal name and a next created arc internal name, where each may range between

1 and 4095. They are both set to 1 by the initialization command, and there are commands to set each of them to a particular value. The only time a created internal name is used is when a vertex or arc is created. If the "CRNAME" option is specified on a creation command, the current created internal name for that type of entity (vertex or arc) is used and that name is then incremented by 1. In the special case where the name "4095" was just used as a created internal name, that name is then set to 1.

4.3.5 Existence and Display

Although the objective of DOGGIE is the display of graphs, it is rather convenient to have vertices and arcs which exist; yet are not necessarily displayed. When a vertex or arc is created, it is not displayed. A separate command is required to display the entity. When a text label is created, it is not displayed either. A separate command must be given to set the display status of a label, which has an effect only if the label exists. If the display status of a label has been set, that label is actually displayed only when the vertex or arc with which it is associated is displayed.

The facilities are available to start or stop the display of any entity in the graph. The display of an arc is independent of the display of its associated vertices, and the display of a vertex is independent of the display of its associated arcs.

Creation and deletion of entities are not independent of the existence or display of other entities. When an entity is created, its internal name must be given. If there is the same type of entity (vertex or arc) with that same name already in existence, it is deleted before the new one is created. The creation of an entity does not

start the display of anything.

Deletion of an arc also stops the display of that arc. In addition to stopping its own display, deletion of a vertex also stops the display of any arc connected to it.

The display of an entity means that it is "drawn" on the paper. Whether or not any or all of a particular displayed entity is visible depends on the particular window size and location.

4.3.6 Intensity, Blinking, Dimming

There are eight-intensity levels (0-7) available for the overall picture intensity. The lowest level (level 0) is barely visible in a dark room, and level 7 is somewhat bright. Unless otherwise changed, DOGGIE uses intensity level 5.

For the purposes of calling attention to some part of the displayed graph, there are two exclusive modes which can be used. When DOGGIE is in BLINK mode any part of the graph may be set to blink at about two cycles per second. Alternatively, when it is in DIM mode, then each part of the graph may be displayed at a chosen intensity level. The parts of the graph which can be individually blinked and dimmed are: vertices, arcs, arrows, and text labels. Once a vertex or arc has been created, its blink or dim status for each part is independent of the existence of display of that part. For example, an arc may be created with an arrow, and then its arrow and label made to blink. Then suppose that a text label is created for the arc. Finally, when the display of the entire arc is requested, the arc itself is displayed steady, and its associated arrow and text label are blinking.

There are commands to set DOGGIE into one mode or the other, and a class of commands to affect the blink or dim status of any part of

any existing vertex or arc.

4.3.7 Light Pen

In the Interactive Graph Theory System the light pen is an important input device used for two distinct functions: inputting screen coordinate information and pointing at displayed vertices and arcs. DOGGIE performs much of the work associated with these two functions.

4.3.7.1 Input and Tracking

Light pen input is included in the Interactive Graph Theory System for both the drawing and the alteration of the layout of a graph. The user has the facility to input this positional information through the location (screen coordinate) of a special bright dot called the "Pseudo-Pen-Point". The name is meant to distinguish this point from the location of the tracking box or cursor (pen-point), which is used to follow the pen. In some graphic systems, the pen-point is the same as the pseudo-pen-point. Here, however, they are kept distinct to provide more flexibility for the user. Pushbuttons provide him with the control over constraining the pseudo-pen-point horizontally, vertically, or both (fixed); meanwhile he may move the pen (and the cursor) anywhere on the screen - the constraints only apply to the pseudo-pen-point.

When neither constraint is in effect the pseudo-pen-point moves in direct correspondence with the cursor, except the pseudo-pen-point will not cross over the screen edge. Similarly, the cursor never crosses over the screen edge. By invoking both the horizontal and vertical constraints, the user may readjust the position of the cursor relative to the pseudo-pen-point.

The tracking as described above is a built-in function of DOGGIE. In addition, there are DOGGIE commands to turn on and off the cursor and pseudo-pen-point, to set the location of the pseudo-pen-point, and to set the location of the cursor. In a graphic system where a tablet is used rather than a pen, the latter command would be meaningless.

4.3.7.2 Pointing

The Interactive Graph Theory System permits the user to select particular vertices or arcs of the graph by pointing at them. Through DOGGIE commands any vertex or arc may be made sensitive to being "hit" by the light pen by enabling the light pen status of that entity. When this status has been enabled and the displayed entity is within the field of view of the pen - usually about one-half of an inch circle - the DOGGIE Light Pen Handler will make note of the first entity which caused the pen to "see" light (according to internal name). Text labels are not sensitive to light pen hits. Note that the handler does not indicate which part of a vertex or arc caused a hit. If the hit entity is a vertex the handler records the screen coordinates of the vertex; if it's an arc, the handler records the screen coordinates of the point where the arc caused the light pen hit. These outputs of the Light Pen Handler are placed into communication cells so that programs may utilize the information.

When a hit occurs, the Light Pen Handler sets an interlock which causes further hits to be ignored until it is cleared by program control.

4.3.8 Pushbuttons

There is a box of pushbuttons next to the display screen which serve as another input mechanism for the user. Three of the buttons are reserved for internal system use and three buttons are used for

control of light pen tracking. The six remaining buttons have no permanent functions, but they may be used for appropriate input aids to suit the needs of each user program. A design philosophy in the Interactive Graph Theory System has been the preference for light buttons as a means of input of user choices. However, there are occasions during user interaction when selective pointing is required at a graph which may appear anywhere on the screen. In such a situation a light button cannot be used since it might be within the field of view of the light pen concurrently with a sensitive part of the graph. The user can perform effective selection by coordinating use of the light pen in one hand, while using his other hand to push a button.

4.3.9 Status of the Graph

There is one type of DOGGIE command used to determine the status of the existing graph. It can be used to find out all information about a particular vertex or arc and can also be used to determine the status of all vertices or arcs, one at a time.

The status information about each entity requested is placed into communication cells so that a program may then read out any desired information. The information is coded compactly in a way which is appropriate for transmission over the telephone line to the central computer.

The status information for a vertex includes the following:

a) Whether the prevailing mode is BLINK mode or DIM mode.

b) If in BLINK mode, the blink status of the vertex shape.

c) If in DIM mode, the blink status of the label.

d) The display status of the vertex.

e) Whether there is a label, and if so, the offset and the text.

f) If there is a label, the display status of the label.

g) Whether the vertex is subject to light pen hits.

h) An indication that the entity is a vertex.

i) The shape of the vertex.

j) The internal name of the vertex.

k) The y-coordinate of the paper position of the vertex.

l) The x-coordinate of the paper position of the vertex.

m) If in DIM mode, the intensity level of the vertex shape.

n) If in DIM mode, the intensity level of the label.

The status information for an arc includes the following:

a) Whether the prevailing mode is BLINK mode or DIM mode.

b) If in BLINK mode, the blink status of the arc itself.

c) If in BLINK mode, the blink status of the arc's arrow.

d) If in BLINK mode, the blink status of the label.

e) The display status of the arc.

f) Whether there is a label, and if so, the offset and the text.

g) If there is a label, the display status of the label.

h) Whether the arc is subject to light pen hits.

i) If the arc is a loop, its orientation.

j) Whether the arc has an arrow.

k) The internal name of the arc.

l) The internal name of the vertex to which the arc is incident.

m) The internal name of the vertex from which the arc is incident.

n) If in DIM mode, the intensity level of the arc itself.

o) If in DIM mode, the intensity level of the arrow.

p) If in DIM mode, the intensity level of the label.

4.4 DOGGIE Command Language

This section presents the pure form of the symbolic DOGGIE Language which is the basis for use in both the IBM 7040 along with ALLA and in the DEC-338 within user programs to affect the display of graphs at the DEC-338. As explained in Section 4.2, the interactive modes of the two environments are realized by the use of communication cells. These are described in Sections 4.6 and 4.7. This section describes what can be represented; the syntax and semantics are given for all commands. The symbolic forms represent sequences of 12-bit words which constitute the machine language of the DOGGIE interpreter. Section 4.5 contains the key to the encoding of the language into the corresponding bit representations.

In the presentation of syntax, braces are used to indicate any one of the entries found within the braces may be used. A pair of brackets is used to enclose an optional language constituent. The terminal words of the language are capitalized, and non-terminals are lower case letters or words. Command strings are generally written in prefix form where various command words and operand words are optional.

Commands are grouped into three classes. The miscellaneous commands refer to those commands which control the gross aspects of the existing graph or the general facilities of DOGGIE including some input/output functions. The commands which control the existence and display of the vertices and arcs of the graph form the important class of commands. The third class includes one command type which is used to determine

the status of the existing vertices and arcs of the graph.

4.4.1 Miscellaneous Commands

There are 16 types of miscellaneous commands which control the gross aspects of the existing graph or the general facilities of DOGGIE.

4.4.1.1 Initialization

Syntax: RESET

Semantics: DOGGIE is initialized as follows:

1. Display of the current graph is stopped.
2. The current graph is entirely deleted, and available storage is allocated.
3. The overall picture intensity is set to level 5.
4. The prevailing mode is made BLINK mode.
5. The window size is set to HALF paper.
6. The location of the cursor (tracking box) and pseudo-pen-point are both set to the center of the screen.
7. The created vertex internal name and created arc internal name are both set to 1.
8. The communication cell GETDAT is restored to its normal contents so that further calls on DOGGIE cause data words to be read in the usual manner. (See Section 4.6.1.1)

4.4.1.2 Intensity

Syntax: SETINT level

where level is an octal digit.

Semantics: The overall picture intensity is set to level.

4.4.1.3 BLINK Mode

Syntax: BLINKM

Semantics: Any blinking or dimming of all vertices and arcs is stopped. The intensity level of all parts of the displayed graph becomes the overall picture intensity level. The prevailing mode is made BLINK mode.

4.4.1.4 DIM Mode

Syntax: DIMM level

where <u>level</u> is an octal digit.

Semantics: Any blinking of all vertices and arcs is stopped. The intensity level of all parts of the existing graph is set to <u>level</u>. The overall picture intensity level is not affected, but it has no effect until the next time BLINK mode is established. The prevailing mode is made DIM mode.

4.4.1.5 Window Size

Syntax: SETWIN { FULL | HALF | FOURTH | EIGHTH }

Semantics: The window size is set to one of the four possible sizes according to the specified option. The position of the window on the paper is unchanged unless it would otherwise overlap the paper's edge. In this case, the window is repositioned at the edge of the paper with a minimum change in window position.

4.4.1.6 Window Position

Syntax: POSWIN, y, x

where <u>y</u> and <u>x</u> are Coordinate Data (See Section 4.4.4).

Semantics: The position of the window on the paper is set to (<u>y</u>,<u>x</u>) which is the y-position and x-position of the paper coordinates of the center of the window unless this positioning would cause the window to overlap the paper's edge. In this case, the window is repositioned at the edge

of the paper as closely as possible to the specified position.

4.4.1.7 Window Movement

Syntax: MOVWIN, dy, dx

where dy and dx are integers between -1023 and +1023.

Semantics: The position of the window on the paper is changed by dy (delta-y) and dx (delta-x) paper units unless this positioning would cause the window to not be fully on the paper. In this case, the window is repositioned at the edge of the paper as closely as possible to the attempted position.

4.4.1.8 Pseudo-Pen-Point Position

Syntax: PSEUDO, y, x

where y and x are Coordinate Data (See Section 4.4.4).

Semantics: The position of the pseudo-pen-point on the window (screen) is set to (y,x) which is the y-position and x-position of the screen coordinates of the point.

4.4.1.9 Cursor Position

Syntax: SETCUR, y, x

where y and x are Coordinate Data (See Section 4.4.4).

Semantics: The position of the cursor (tracking box) on the window (screen) is set to (y,x) which is the y-position and x-position of the screen coordinates of the center of the cursor. If this positioning would cause the cursor to overlap the window's edge, then the cursor is repositioned at the edge of the window as closely as possible to the specified position.

4.4.1.10 Light Pen Tracking

Syntax: $\left\{\begin{matrix}\text{START}\\ \text{STOP}\end{matrix}\right\}$ CURSOR

Semantics: The "START" option causes the cursor (tracking box) and pseudo-pen-point to be displayed by setting the pushbutton which the user also has available for this function. Tracking may then be performed. This option also clears communication cell CHPSEU, which is used to indicate a change in the location of the pseudo-pen-point due to tracking by the light pen (see Sections 4.6.2.6 and 4.7.4.6).

The "STOP" option stops the display of both the cursor (tracking box) and pseudo-pen-point by clearing the pushbutton which the user also has available for this function.

4.4.1.11 Light Pen Hits

Syntax: ALLHIT

Semantics: The light pen hit interlock (communication cell LPHIT1) is cleared so that the Light Pen Handler will interpret the next light pen hit. The communication cell LPHIT2 is also cleared.

4.4.1.12 Created Internal Names

Syntax: SETCRN $\left\{\begin{matrix}\text{VERTEX}\\ \text{ARC}\end{matrix}\right\}$, name

where name is an integer between 1 and 4095.

Semantics: The created vertex internal name or the created arc internal name is set to name, according to the specified option.

4.4.1.13 Pushbutton Clearing

Syntax: CLRPB, nnnn

where nnnn is some configuration of 12 bits.

Semantics: Those pushbuttons corresponding to the bits of nnnn which are ZERO's are cleared. Bit 0 of nnnn corresponds to pushbutton 0, bit

1 corresponds to pushbutton 1, etc.

4.4.1.14 Pushbutton Setting

Syntax: SETPB, nnnn

where nnnn is some configuration of 12 bits.

Semantics: Those pushbuttons corresponding to the bits of nnnn which are ONE's are set. Bit 0 of nnnn corresponds to pushbutton 0, bit 1 corresponds to pushbutton 1, etc.

4.4.1.15 Teletype Output

Syntax: TYPE, codes

where codes is a string of 8-bit character codes.

Semantics: Each of the given character codes in codes is outputted on the Teletype. ASCII character codes may be used for typed text. However, this command may also be used to punch any sequence of 8-bit codes on the Teletype punch of the DEC-338.

4.4.1.16 Loading Program Segments

Syntax: { LOAD , segnam / GOTO field , addr / LOADGO field, segnam, addr }

```
         ⎧ LOAD         ,   segnam       ⎫
Syntax:  ⎨ GOTO field   ,   addr         ⎬
         ⎩ LOADGO field,    segnam, addr ⎭
```

Semantics: When the LOAD option is used, the program segment whose name (1 to 4 characters) is segnam is loaded into the memory of the DEC-338 from the disk.

The GOTO option causes DOGGIE to stop interpreting, and PDP-8 control is transferred to the DEC-338 location addr of memory field field. This option also causes the communication cell GETDAT to be restored to its normal contents so that further calls on the DOGGIE interpreter cause data words to be read in the usual manner. (See Section 4.6.1.1)

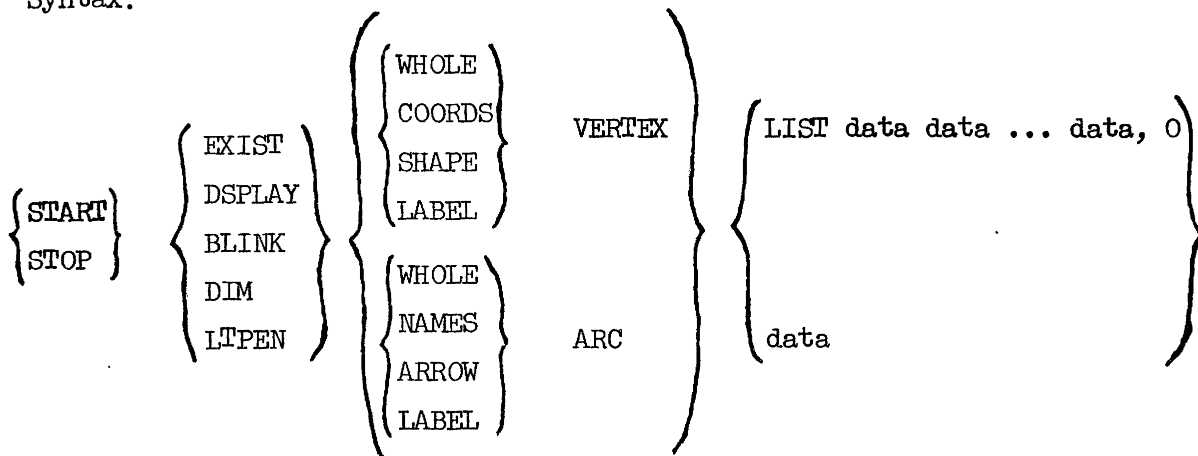
The LOADGO option is a combination of the above two options. When this is used, the segment is first completely loaded, and then control is transferred. The command words causing this loading may be in an area which is overlayed by the loading segment.

(The programmer must beware that use of either the LOAD or LOADGO options will not preserve the contents of locations 7200_8 - 7577_8 of the DEC-338.)

4.4.2 Graph Commands

This class of commands is used to effect changes in the existence, display, and light pen status of the graph maintained in the DEC-338.

Syntax:



The following subsections include the semantics of possible groupings of these command words and with each section is included the syntax and semantics for "data." data may include "name", which is an integer between 1 and 4095.

The LIST option provides a shorter form for the use of the same fundamental command type with a list of arguments. For example, this option is applicable when one wishes to start a list of ten vertices blinking. Ten individual commands can be used, but one command with a list of ten vertex internal names is more concise. A specific example

is given in Section 4.7.1.

For explanatory purposes the graph commands are presented with only one set of data arguments. All groupings can be used with the LIST option; however, note that a name of zero terminates the list.

4.4.2.1 Creating a Vertex

Syntax: START EXIST WHOLE VERTEX shape, $\left\{\begin{matrix}\text{name}\\ \text{CRNAME}\end{matrix}\right\}$, y,x

where <u>shape</u> is an octal digit, and

where <u>y</u> and <u>x</u> are Coordinate Data (see Section 4.4.4).

Semantics: If the "CRNAME" option is specified, a created vertex internal name is used. Any vertex with the given internal name is first deleted (see Section 4.3.5), and then a new vertex with the given internal name is created at the paper location specified by <u>y</u> and <u>x</u>. The shape of this vertex is set according to the given octal digit for <u>shape</u>. The vertex is created with light pen status disabled and with no label. If the prevailing mode is BLINK mode, all parts of the vertex are initialized to be not blinking. If it is DIM mode, all parts are initialized to be at the dimming intensity level. This command merely defines a vertex, but does not start its display.

4.4.2.2 Creating an Arc

Syntax:

START EXIST WHOLE ARC [ARROWD] $\left[\left\{\begin{matrix}\text{LOOPE}\\ \text{LOOPN}\\ \text{LOOPW}\\ \text{LOOPS}\end{matrix}\right\}\right]$,$\left\{\begin{matrix}\text{name}\\ \text{CRNAME}\end{matrix}\right\}$, toname, fromname

where <u>toname</u> and <u>fromname</u> are integers between 1 and 4095.

Semantics: If the "CRNAME" option is specified, a created arc internal name is used. Any arc with the given internal name is first deleted (see Section 4.3.5), and then a new arc with the given internal name

is defined in terms of the vertices to which and from which it is incident. These are given by the vertex internal names toname and fromname, respectively; and these internal names may be the same, thereby defining a loop. If a loop is specified, either the command may include a specific orientation East, North, West, South (according to LOOPE, LOOPN, LOOPW, LOOPS), or an East loop will be used as a default case. The arc will be created with an arrow if the "ARROWD" option is included in the command. The arc is created with light pen status disabled and with no label. If the prevailing mode is BLINK mode, all parts of the arc are initialized to be not blinking. If it is DIM mode, all parts are initialized to be at the dimming intensity level. This command merely defines an arc, but does not start its display. Note that the existence of the vertices toname and fromname is not required at the time when the arc is defined.

4.4.2.3 Deleting a Vertex

Syntax: STOP EXIST WHOLE VERTEX, $\begin{Bmatrix}\text{name}\\\text{ALL}\end{Bmatrix}$

Semantics: The vertex whose internal name is name (including its label) is deleted from the graph. If such a vertex does not already exist, the entire command is ignored. If the "ALL" option is specified, all vertices are deleted. The display of each deleted vertex is immediately stopped, and all information about the vertex is lost. In addition, the display is stopped for each arc which is defined in terms of any deleted vertex (either incident to or incident from the vertex).

4.4.2.4 Deleting an Arc

Syntax: STOP EXIST WHOLE ARC, $\begin{Bmatrix}\text{name}\\\text{ALL}\end{Bmatrix}$

Semantics: The arc whose internal name is name (including its label)

is deleted from the graph. If such an arc does not already exist, the entire command is ignored. If the "ALL" option is specified, all arcs are deleted. The display of each deleted arc is immediately stopped, and all information about the arc is lost.

4.4.2.5 Altering a Vertex Position

Note: It is considered meaningless to "STOP EXIST COORDS VERTEX..." and therefore such commands are ignored.

Syntax: START EXIST COORDS VERTEX, name, y, x

where y and x are Coordinate Data (see Section 4.4.4)

Semantics: The vertex whose internal name is name is repositioned at the paper position specified by y and x. If such a vertex does not exist, the entire command is ignored. This command does not affect the display status of the vertex. All displayed arcs defined in terms of this vertex are appropriately updated so that they all remain attached to the vertex.

4.4.2.6 Altering a Vertex Shape

Syntax: START EXIST SHAPE VERTEX shape, $\begin{Bmatrix}\text{name} \\ \text{ALL}\end{Bmatrix}$

or

STOP EXIST SHAPE VERTEX, $\begin{Bmatrix}\text{name} \\ \text{ALL}\end{Bmatrix}$

where shape is an octal digit.

Semantics: The "START" version of this command sets the shape of vertex whose internal name is name to one of the eight possible shapes according to the given octal digit for shape. The "STOP" version of the command sets the shape of the specified vertex to zero. If such a vertex does not exist, the entire command is ignored. If the "ALL" option is specified, the operation is performed on all vertices. This command does not affect the display status of the vertex; however, if the vertex

is being displayed the change of shape will be immediately observable.

4.4.2.7 Altering the Vertex Names of an Arc

Note: It is considered meaningless to "STOP EXIST NAMES ARC...", and therefore such commands are ignored.

Syntax:

$$\text{START EXIST NAMES ARC} \left[\text{LOOP} \begin{Bmatrix} \text{LOOPE} \\ \text{LOOPN} \\ \text{LOOPW} \\ \text{LOOPS} \end{Bmatrix} \right] \text{, name, toname, fromname}$$

where toname and fromname are integers between 1 and 4095.

Semantics: The arc whose internal name is name is redefined to be incident to vertex toname and incident from vertex fromname. If such an arc does not exist, the entire command is ignored. If the two specified vertex internal names are the same, a loop is defined, in which case either the command may include the "LOOP" option along with a specific orientation or an East loop will be used as a default case. This command does not affect the display status of the arc unless either vertex toname or vertex fromname does not exist, in which case, the arc cannot be displayed.

4.4.2.8 Creating an Arrow (and Altering Loop Orientation)

Syntax: $$\text{START EXIST ARROW ARC} \left[\text{LOOP} \begin{Bmatrix} \text{LOOPE} \\ \text{LOOPN} \\ \text{LOOPW} \\ \text{LOOPS} \end{Bmatrix} \right] \text{,} \begin{Bmatrix} \text{name} \\ \text{ALL} \end{Bmatrix}$$

Semantics: An arrow is placed on the arc whose internal name is name. If such an arc does not exist, the entire command is ignored. If the "ALL" option is specified, the operation is performed on all arcs. If the affected arc is a loop, the command may also be used with the "LOOP"

option to reset the loop orientation to either East, North, West, or South according to one of the options LOOPE, LOOPN, LOOPW, or LOOPS. This command does not affect the display status of the arc.

4.4.2.9 Deleting an Arrow

Syntax: STOP EXIST ARROW ARC, {name / ALL}

Semantics: An arrow is deleted from the arc whose internal name is name. If such an arc does not exist, the entire command is ignored. If the "ALL" option is specified, the arrows of all arcs are deleted. This command does not affect the display status of the arc.

4.4.2.10 Creating and Altering a Label

Syntax:

START EXIST LABEL {VERTEX / ARC} [{OFFSET / DEFALT}] [TEXT],name[,offy,offx][, @chars@]

where offy and offx must be given if and only if "OFFSET" is included; each may be an integer between 0 and 1023 to specify a positive offset; a negative offset is specified by the word "MINUS" followed by integer between 0 and 1023.

where chars must be given if and only if "TEXT" is included; it is a list (possibly null) of text characters.

Semantics: This command is used to create or alter a label for either the vertex or arc whose internal name is name. If such an entity does not exist, the entire command is ignored.

If the entity did not previously have a label, the command creates a label with text (chars), or with a position offset (offy and offx), or both. If neither of these options is used, a null label is still created. If the "OFFSET" option is not used, a default offset is assumed. When a label is first created, its text is not displayed.

If the label for the entity already existed, this command either replaces the text by <u>chars</u>, or sets the position offset to either default values or <u>offy</u> and <u>offx</u>, or both, or neither. If neither the "OFFSET" or "DEFALT" option is given, the offset is unchanged. Similarly, if the "TEXT" option is not given, the text of the label is unchanged. This command does not affect the display status of either the entity or the label.

4.4.2.11 Deleting a Label

Syntax: $\text{STOP EXIST LABEL} \left\{\begin{matrix}\text{VERTEX}\\ \text{ARC}\end{matrix}\right\}, \left\{\begin{matrix}\text{name}\\ \text{ALL}\end{matrix}\right\}$

Semantics: The label (both text and offset) associated with the vertex or arc whose internal name is <u>name</u> is deleted as if it never existed. If such an entity does not exist, the entire command is ignored. If the "ALL" option is given, the operation is performed on the labels of all vertices or the labels of all arcs, according to which type of entity is specified. The display of any deleted label is immediately stopped.

4.4.2.12 Display Control

$$\text{Syntax:} \left\{\begin{matrix}\text{START}\\ \text{STOP}\end{matrix}\right\} \text{ DSPLAY } \left\{\begin{matrix}\left\{\begin{matrix}\text{WHOLE}\\ \text{SHAPE}\\ \text{LABEL}\end{matrix}\right\} & \text{VERTEX}\\ \left\{\begin{matrix}\text{WHOLE}\\ \text{NAMES}\\ \text{ARROW}\\ \text{LABEL}\end{matrix}\right\} & \text{ARC}\end{matrix}\right\}, \left\{\begin{matrix}\text{name}\\ \text{ALL}\end{matrix}\right\}$$

Semantics: A command of this form starts or stops the display of either one part or all parts of either the vertex or arc whose internal name is <u>name</u>, according to the specified options. If such an entity does not exist, the entire command is ignored. If the "ALL" option is specified,

the operation is performed on either all vertices or all arcs, according to the specified option for the type of entity.

The "WHOLE" option for the display of a vertex is used to affect the display of the vertex shape and label (if the label exists).

The "NAMES" option for the display of an arc is used to affect the display of the arc itself. The "WHOLE" option affects the display of the arc itself, the arrow, and the label (if the label exists).

An arc may be displayed as a result of an appropriate display command only if both its from-vertex and to-vertex exist; otherwise, the command has no effect.

4.4.2.13 Blinking

Syntax:
$$\left\{\begin{array}{l}\text{START}\\\text{STOP}\end{array}\right\}\ \text{BLINK}\ \left\{\begin{array}{ll}\left\{\begin{array}{l}\text{WHOLE}\\\text{SHAPE}\\\text{LABEL}\end{array}\right\} & \text{VERTEX}\\ \left\{\begin{array}{l}\text{WHOLE}\\\text{NAMES}\\\text{ARROW}\\\text{LABEL}\end{array}\right\} & \text{ARC}\end{array}\right\},\ \left\{\begin{array}{l}\text{name}\\\text{ALL}\end{array}\right\}$$

Semantics: A command of this form may be used only when the prevailing mode is BLINK mode. The blinking of either one part or all parts of either the vertex or arc whose internal name is <u>name</u>, is started or stopped according to the specified options. If such an entity does not exist, the entire command is ignored. If the "ALL" option is specified; the operation is performed on either all vertices or all arcs, according to the specified option for the type of entity.

The "WHOLE" option for the blinking of a vertex is used to affect the blink status of the vertex shape and label (if the label exists).

The "NAMES" option for the blinking of an arc is used to affect the blink status of the arc itself. The "WHOLE" option affects the arc itself, the arrow, and the label (if the label exists).

4.4.2.14 Dimming

$$\text{Syntax: } \left\{\begin{matrix}\text{START}\\ \text{STOP}\end{matrix}\right\} \text{ DIM } \left\{\begin{matrix} \left\{\begin{matrix}\text{WHOLE}\\ \text{SHAPE}\\ \text{LABEL}\end{matrix}\right\} & \text{VERTEX} \\ \left\{\begin{matrix}\text{WHOLE}\\ \text{NAMES}\\ \text{ARROW}\\ \text{LABEL}\end{matrix}\right\} & \text{ARC}\end{matrix}\right\} \text{ level, } \left\{\begin{matrix}\text{name}\\ \text{ALL}\end{matrix}\right\}$$

where level must be given if the "START" option is given; it is an octal digit.

Semantics: A command of this form may be used only when the prevailing mode is DIM mode. The intensity level of either one part or all parts of either the vertex or arc whose internal name is name is set, according to the specified options. If such an entity does not exist, the entire command is ignored. If the "ALL" option is specified, the operation is performed on either all vertices or all arcs, according to the specified option for the type of entity.

The "START" option is used to set the intensity level of the selected part of the graph to level. The "STOP" option causes the intensity level of the selected part of the graph to be set to the current dimming level according to the most recent DIMM command.

The "WHOLE" option for the dimming of a vertex is used to affect the intensity level of the vertex shape and label (if the label exists).

The "NAMES" option for the dimming of an arc is used to affect the intensity level of the arc itself. The "WHOLE" option affects the arc

itself, the arrow, and the label (if the label exists).

4.4.2.15 Light Pen Status

Syntax: $\begin{Bmatrix} \text{START} \\ \text{STOP} \end{Bmatrix}$ LTPEN WHOLE $\begin{Bmatrix} \text{VERTEX} \\ \text{ARC} \end{Bmatrix}$, $\begin{Bmatrix} \text{name} \\ \text{ALL} \end{Bmatrix}$

Semantics: The light pen status for the vertex or arc whose internal name is name is enabled ("START") or disabled ("STOP"), according to the specified options. If such an entity does not exist, the entire command is ignored. If the "ALL" option is specified, the operation is performed on either all vertices or all arcs, according to the specified option for the type of entity. When light pen status is enabled on an entity, it is subject to being "observed" or "hit" by the light pen (see Section 4.3.7.2).

4.4.3 Graph Status

Syntax: [START] STATUS $\begin{Bmatrix} \text{VERTEX} \\ \text{ARC} \end{Bmatrix}$, $\begin{Bmatrix} \text{name} \\ \text{NEXT} \end{Bmatrix}$

Semantics: When the "START" or "NEXT" options are not specified, status information is placed into the status communication cells for either the vertex name or the arc name, according to the specified option. If such an entity does not exist, one of the communication cells is set to an indicative value.

The "START" and "NEXT" options are available for the interrogation of the existing graph when the name of the entity is not known. They may be used, for example, to read out the entire existing graph.

When the "START" option is specified, the argument given as "name" or "NEXT" is ignored. Status of the vertex or arc (according to the specified option) most recently created is placed into the status communication cells. If there are no existing entities of the specified

type, one of the communication cells is set to an indicative value.

When the "START" option is not specified, but the "NEXT" option is, the status of the next younger vertex or arc (according to the specified option) in the graph is placed into the status communication cells. "Next younger" refers to the last entity of that type whose status was requested. The "NEXT" option may not be used if the last entity whose status was examined has since been deleted (including deletion due to initialization). If there is no next younger entity of the specified type (i.e., if the last examined entity was the youngest), then one of the communication cells is set to an indicative value.

4.4.4 Coordinate Data

The following types of commands have a pair of arguments which specify coordinate data:

POSWIN...

PSEUDO...

SETCUR...

START EXIST WHOLE VERTEX...

START EXIST COORDS VERTEX...

The syntactic variables used in the description of these five types are "y" and "x". The syntax and semantics for y (or x) is given below.

Syntax: [PENPNT] [SCREEN] size

where size is an integer between 0 and 1023.

Semantics: The value of the given expression is an integer between 0 and 1023. This value is used as a screen coordinate for the PSEUDO and SETCUR commands and as a paper coordinate for the others.

If neither option is specified, the value of the data is simply the value of size.

If only the "PENPNT" option is specified, the value of the data is the value of the y (or x) screen coordinate of the position of the pseudo-pen-point plus the value of size modulo 1024. This option is usually used with size = 0.

If only the "SCREEN" option is specified, the value of the data is the value of size transformed (according to the current window size) to the y (or x) paper coordinate position closest to the corresponding y (or x) screen coordinate position.

If both the "PENPNT" and "SCREEN" options are specified, an intermediate value is first determined, and that value is then transformed. The intermediate value is the value of the y (or x) screen coordinate of the position of the pseudo-pen-point plus the value of size modulo 1024. The final value of the data is the intermediate value transformed (according to the current window size) to the y (or x) paper coordinate position closest to the corresponding y (or x) screen coordinate position.

4.5 Encoding of the DOGGIE Command Language

The preceding section fully presented the pure form of the symbolic DOGGIE Language. The symbolic statements merely represent sequences of 12-bit words which constitute the machine language of the DOGGIE interpreter. This section describes the encoding of the DOGGIE statements into these 12-bit quantities.

Each comma in a DOGGIE symbolic statement separates 12-bit words except when it is used between two at-signs as a text label constituent. Each 12-bit word may be represented by one or more terms separated by spaces where each term has a corresponding value, and the value of a 12-

bit word is the inclusive-OR of the values of its terms. If a term is already a number, such as the shape specification for a vertex, the numeric value of the number is the term's value. Table 4-2 gives the corresponding values of each DOGGIE word which may appear as a term within a DOGGIE statement. The table is in alphabetical order.

Although Table 4-2 provides most of the information necessary for encoding statements of the DOGGIE Language, the following subsections include further information pertinent to the cases not yet covered.

4.5.1 MOVWIN Command

dy and dx are given as 12-bit two's complement integers.

4.5.2 Offset of a Label

offy and offx are given as 11-bit sign-magnitude integers, where the high order bit of the 12-bit word is ignored and the next bit is the sign bit.

4.5.3 Text Characters of a Label

chars represents a string of 6-bit trimmed ASCII character codes. Two characters per 12-bit word are encoded according to Table 4-3, where the high-order half of the word contains the first character of the pair. The list terminates with a character code of 00, and if this code appears as the first character of a pair, the second character code is also 00.

4.5.4 Name of a Program Segment

segnam represents four characters given as two 12-bit words packed two characters per word. Each character is coded according to its trimmed ASCII character code given in Table 4-3. The high-order half of each word contains the first character of the pair. Names of less than four characters are padded with the character 00.

Table 4-2 DOGGIE Words and Their Values

Word	Value (octal)
ALL	0000
ALLHIT	0260
ARC	0000
ARROW	0040
ARROWD	0001
BLINK	2400
BLINKM	0160
CLRPB	0300
COORDS	0020
CRNAME	0000
CURSOR	0240
DEFALT	0002
DIM	2400
DIMM	1160
DSPLAY	2200
EIGHTH	0003
EXIST	2000
FOURTH	0002
FULL	0000
GOTO	1340
HALF	0001
LABEL	0060
LIST	0100
LOAD	0350
LOADGO	1350
LOOP	0001
LOOPE	0000
LOOPN	0002
LOOPS	0006
LOOPW	0004
LTPEN	2600
MINUS	2000
MOVWIN	0100
NAMES	0020
NEXT	0000
OFFSET	0004
PENPNT	4000
POSWIN	0060
PSEUDO	0120
RESET	0360
SCREEN	2000
SETCRN	0200
SETCUR	0140
SETINT	0020
SETPB	1300
SETWIN	0040
SHAPE	0040
START	1000
STATUS	0220
STOP	0000
TEXT	0001
TYPE	0320
VERTEX	0010
WHOLE	0000

Table 4-3 Trimmed ASCII Character Codes

Character	Code Value (octal)	Character	Code Value (octal)
end-of-list	00	space	40
A	01	!	41
B	02	"	42
C	03	#	43
D	04	$	44
E	05	%	45
F	06	&	46
G	07	'	47
H	10	(	50
I	11	)	51
J	12	*	52
K	13	+	53
L	14	,	54
M	15	-	55
N	16	.	56
O	17	/	57
P	20	0	60
Q	21	1	61
R	22	2	62
S	23	3	63
T	24	4	64
U	25	5	65
V	26	6	66
W	27	7	67
X	30	8	70
Y	31	9	71
Z	32	:	72
[	33	;	73
\	34	<	74
]	35	=	75
↑	36	>	76
←	37	?	77

4.5.5 TYPE Command

<u>codes</u> is given as a string of 12-bit words, ending with a terminator word of all ONES. Bits 4-11 of each code word are used as the 8-bit outputted code, and the other bits are ignored.

4.6 Interactive Programs in the DEC-338

The design of DOGGIE was based primarily on the goal of producing an executive which provided the environment for easily writing effective interactive graphical programs which operate locally at the DEC-338. The chosen organization also had to support interactive programs between the central computer and the terminal, and thus the use of an interpreter in the DEC-338 was rather helpful. The DOGGIE interpreter accepts sequences of 12-bit words as commands. During the operation of local user programs, these commands are directly scanned as a list in the memory of the DEC-338. In the course of interaction with the central computer, these commands are received over the Dataphone and directly scanned by the interpreter. In either case, the commands are those described in Section 4.4 - the DOGGIE Command Language.

One possible design would have been to augment DOGGIE with those operations which could make it into a programming language. Then local user programs would be entirely interpreted. The author felt this approach would yield an ineffective implementation in the DEC-338. One must keep in mind 8K of 12-bits with a rather primitive set of instructions (i.e. the DEC-338) is very limiting in the design of an elaborate system. Enough power had to be kept in the interpreted language for the central computer to control the terminal. By including the GOTO command in the DOGGIE Language, the central computer can cause a specific

function to be performed at the terminal. Since that function may be of arbitrary complexity, and may include effective interaction, there is no need for interactive statements to be in the DOGGIE Language itself. This type of statement would be, for example, one which specified a particular operation to be performed as a result of a particular action of the user.

The extension of the DOGGIE Language into a programming language has been done by imbedding it into a programming language environment. In the DEC-338 this was done by allowing programs the power of the PDP-8 machine. In fact, local user programs of the Interactive Graph Theory System are primarily PDP-8 programs which include calls upon DOGGIE both as an interpreter and an executive whenever necessary. The DOGGIE interpreter is treated as a PDP-8 subroutine called by a standard PDP-8 subroutine call.

The interactive components of local user programs are made possible by the use of communication cells on the initial page of field 0. This concept was introduced in Section 4.2, where it was also mentioned the same philosophy is used in the interactive programs operating in the central computer. In the DEC-338 there are several of these communication cells, merely PDP-8 locations, dedicated to the interfacing of user programs with DOGGIE. In fact all interfacing is done via these calls. When the user programs are written using the PDPMAP Assembly System, the programmer refers to the cells by symbolic name, and that is how they will be mentioned throughout this section.

In its current implementation the DEC-338 is allocated by DOGGIE so that locations 6000_8 through 7577_8 are available for execution of

local user programs. These programs are stored on the disk of the DEC-338 as absolute machine language programs. They are called in by name via the DOGGIE commands "LOAD" or "LOADGO", and these commands may be used to bring in overlays of program segments, so there is practically no limit on the size of any user program.

The remainder of this section consists of four subsections which serve as a programmer's description for the writing of user programs in the Interactive Graph Theory System. Section 4.6.1 covers the routines available to the programmer which are part of DOGGIE and available through communication cells. Then, Section 4.6.2 describes those communication cells used for status information concerning all of the functions of DOGGIE. Section 4.6.3 describes the syntax used for DOGGIE commands in user programs, and the development culminates in an example given in Section 4.6.4.

4.6.1 Available Routines

The following subsections present a description of those routines available in DOGGIE which user programs may call via communication cells. The programmer of user programs should be familiar with the information in the first four subsections. The remainder deal with particular input/output functions which many user programs do not require.

4.6.1.1 The Interpreter

The DOGGIE interpreter is a subroutine within the DEC-338 which is pointed to by the contents of the communication cell named DOGGIE. Therefore, the following subroutine call is used to initiate interpretation of a list of DOGGIE commands which start at location Y in the memory of the DEC-338:

```
JMS* DOGGIE
PZE Y-*
```

Note the calling sequence consists of two words where the second one contains the argument which is a relative pointer to the list of DOGGIE words, which may be anywhere within the same memory field as the calling sequence.

The list beginning at location Y may consist of any number of DOGGIE commands, one after the other, in sequential memory locations. The list is terminated by a command of 0000 which takes no arguments. When the interpreter decodes this special terminator word at a time when it expects the beginning of a command, interpretation is suspended, and control is returned to the calling program immediately following the calling sequence.

As just described, the interpreter fetches words from a list starting at location Y. This is, in fact, the normal mode of using the interpreter; however, additional flexibility has been built into the interpreter for special requirements. The interpreter actually fetches its input words by calling a subroutine through a pointer in communication cell GETDAT. In the normal case, this cell points to a subroutine within DOGGIE which feeds the interpreter from the list specified in the call to the interpreter. The DEC-338 programmer may alter the contents of GETDAT in order to substitute his own (alternate) subroutine for getting input data when, for example, a list is an inconvenient form of input. The cell GETDAT may be changed to point to any field 0 location where the programmer wishes to supply an alternate routine. Such a routine may assume a cleared accumulator upon entry.

This option is used when the DEC-338 is set up to interpret DOGGIE commands generated in the central computer. In this case the alternate routine obtains input data for the interpreter from the Dataphone servicing routines.

The contents of cell GETDAT is restored to its normal contents whenever the RESET (see Section 4.4.1.1) or GOTO (see Section 4.4.1.16) commands are executed and whenever the Graph Monitor is restarted (see Section 4.6.2). Also, the communication cell RESGET contains a pointer to the subroutine which restores GETDAT. The following subroutine call may be used to cause the restoration:

```
JMS* RESGET
```

4.6.1.2 Restarting the Graph Monitor

In order for DOGGIE to properly function, it is required there be a program segment named "GMON" on the disk of the DEC-338 which is a program whose entry point is location 6000_8. In the Interactive Graph Theory System this segment is the Graph Monitor which is used to control the system by light buttons. The Graph Monitor is automatically begun when DOGGIE is loaded and started. Also, the convention has been established to return to the Graph Monitor when the user depresses the large button on the pushbutton box - the manual interrupt button. Since this is such a common function, a routine to restart the Graph Monitor is included within DOGGIE. The communication cell RESTRT contains a pointer to this routine. Thus, the following instruction may be used to terminate a user program and restart the Graph Monitor:

```
JMP* RESTRT
```

The restart routine also causes the contents of communication cell GETDAT to be restored to its normal contents so that further calls upon the interpreter cause data words to be read in the normal manner (see Section 4.6.1).

4.6.1.3 Manual Interrupt Button

The Interactive Graph Theory System is used with the convention that depressing the manual interrupt button suspends the current operation, and causes the Graph Monitor to be started. Since not all programs can be arbitrarily interrupted anywhere, this convention was not made an automatic feature of DOGGIE. This has the advantage of possibly abandoning the convention at certain times during console operation. Programs therefore must be written to check for this button's having been pushed.

There is a software flag within DOGGIE, called the Manual Interrupt Flag, which is set whenever a user hits the manual interrupt button on the pushbutton box. The communication cell MANINT contains a pointer to the subroutine which checks the status of the Manual Interrupt Flag. The following subroutine call returns to the next location if the flag is set; if the flag is clear, the subroutine returns by a skip. After the sense of the return is determined, the subroutine clears the Manual Interrupt Flag.

```
JMS* MANINT
return 1          flag set
return 2          flag clear
```

4.6.1.4 Pushbutton Handling

The DOGGIE Command Language includes the CLRPB and SETPB commands for the clearing or setting of one or more of the twelve pushbuttons

on the pushbutton box. A user program may also directly call the basic subroutines in DOGGIE which perform pushbutton clearing or setting. Another subroutine is available which allows the user program to interrogate the status of any one of the pushbuttons.

4.6.1.4.1 Clearing Pushbuttons

The communication cell PBCLR contains a pointer to the subroutine which clears pushbuttons. The following subroutine call is used to clear those pushbuttons corresponding to the bits of the argument "nnnn" which are ZERO's. (Bit 0 of "nnnn" corresponds to pushbutton 0, bit 1 corresponds to pushbutton 1, etc.)

```
JMS*  PBCLR
OCT   nnnn
```

4.6.1.4.2 Setting Pushbuttons

The communication cell PBSET contains a pointer to the subroutine which sets pushbuttons. The following subroutine call is used to set those pushbuttons corresponding to the bits of the argument "nnnn" which are ONE's. (Bit 0 of "nnnn" corresponds to pushbutton 0, bit 1 corresponds to pushbutton 1, etc.)

```
JMS*  PBSET
OCT   nnnn
```

4.6.1.4.3 Pushbutton Status

The communication cell PBSKIP contains a pointer to the subroutine which interrogates pushbutton status. The following subroutine call is used to check those pushbuttons corresponding to the bits of the argument "nnnn" which are ONE's. (Bit 0 of "nnnn" corresponds to pushbutton 0, bit 1 corresponds to pushbutton 1, etc.) The subroutine returns to the location after the argument if all interrogated push-

buttons are ZERO. If any of the interrogated pushbuttons are ONE the subroutine returns by a skip to two locations after the argument.

```
JMS*  PBSKIP
OCT   nnnn
return 1        all selected pushbuttons are ZERO
return 2        one or more selected pushbuttons are ONE
```

4.6.1.5 Teletype Input/Output

Four communication cells are available for Teletype input and output. DOGGIE maintains a software completion flag and a character buffer for both input and output, which are cleared, set, and checked in the same way that the four corresponding PDP-8 IOT instructions operate with the hardware flags (KSF, KRB, TSF, TLS).

4.6.1.5.1 Teletype Input

There is a Teletype input flag which is set whenever a key is struck on the keyboard. The communication cell KSFKSF contains a pointer to the subroutine which checks the status of the Teletype input flag. The following subroutine call returns to the next location if the flag is clear; if the flag is set, the subroutine returns by a skip. The subroutine preserves the contents of the accumulator and link.

```
JMS*  KSFKSF
return 1        flag clear
return 2        flag set
```

When the Teletype input flag is set, the ASCII character code corresponding to the struck key is assembled in the reader buffer. The communication cell KRBKRB contains a pointer to the subroutine which reads the reader buffer into the accumulator. The following subroutine call causes the contents of the accumulator to be ignored, and to be

replaced with the 8-bit character code in the low-order eight bits and with the high-order four bits set to ZERO. The subroutine also clears the Teletype input flag. The subroutine preserves the contents of the link.

    JMS* KRBKRB

4.6.1.5.2 Teletype Output

The DOGGIE Command Language includes the TYPE command for outputting to the Teletype. A user program may also directly call the basic subroutines in DOGGIE which are used for Teletype output.

There is a Teletype output flag which is set whenever a character may be typed. The flag is clear only during the 100 ms. period after a character has been typed. The programmer may not want to bother checking this flag since the type-out subroutine always waits for the flag to be set (unlike the PDP-8 hardware).

For completeness and for making it possible to overlap output with computation, the facility to test the output flag is available. The communication cell TSFTSF contains a pointer to the subroutine which checks the status of the Teletype output flag. The following subroutine call returns to the next location if the flag is clear; if the flag is set, the subroutine returns by a skip. The subroutine preserves the contents of the accumulator and link.

```
JMS*  TSFTSF
return 1          flag clear
return 2          flag set
```

The communication cell TLSTLS contains a pointer to the subroutine which causes a character to be typed according to the character code contained in the low order eight bits of the accumulator. The following

subroutine call first waits for the output flag to become set, then types the character, clears the output flag, and finally returns with a clear accumulator. The subroutine preserves the contents of the link.

```
JMS*  TLSTLS
```

4.6.1.6 Using the Disk

The communication cell SYSIO contains a pointer to a subroutine equivalent to the standard disk input/output routine which is part of the PDP-8 Disk Monitor System [53]. A user program may include the following calling sequence in order to use the disk as a storage device:

```
JMS*  SYSIO
function (see below)
block to be accessed
low order memory address
(alternate return address)
(normal return here)
```

The above call causes a reading from or writing onto a particular block of the disk. The Disk System treats the disk as 253 blocks of 128 words each. The blocks are numbered from 0000 through 0374_8. Data may be exchanged between memory and disk only as one block at a time. The "function" word controls the operation as follows:

Bits 0 - 1	Unused
Bit 2	If 0, the normal return is used. If 1, ONE plus the address contained in the fourth parameter word is used as a return address.
Bits 3 - 5	Unused
Bits 6 - 8	Memory Field

```
Bits 9 - 11        = 3 for reading from disk
                   = 5 for writing onto disk
```

4.6.1.6.1 Disk Activity Indicator

The communication cell DREADY contains a pointer to the subroutine which checks whether there is a disk operation in progress either due to a call via SYSIO or execution of the LOAD DOGGIE command. If there is, the return from the following subroutine call is to the next location. If there is no disk operation in progress the return from the subroutine skips.

```
JMS* DREADY
return 1        disk active
return 2        disk activity complete
```

4.6.1.7 Dataphone Communications

DOGGIE handles Dataphone Communications through interrupt-time service routines which use the DEC 637 Dataphone Interface. In the Interactive Graph Theory System there are program segments kept on the disk of the DEC-338 which perform buffered Dataphone communications including various timing considerations, checksum computations, etc. These routines call upon the facilities described in this section. The programmer writing user programs need not be familiar with the detailed level presented here. He should use the standard buffering routines as described in a separate memorandum. The information presented here is directed to the systems programmer.

Three communication cells are available for Dataphone communications using DOGGIE. The cell SNDCH contains a pointer to a location used for sending. The cell RCVCH contains a pointer to a location used for receiving. The cell INTRET contains a pointer to the

"interrupt return" location to which control should be transferred after a user function program completes either sending or receiving.

4.6.1.7.1 Sending

The communication cell SNDCH contains a pointer to a location which is used as both an indicator and a subroutine entry point for sending a character. In order to maintain a sense of timing, DOGGIE is always transmitting (sending) to the Dataphone line. Every 3 1/3 ms., the Dataphone transmit flag causes an interrupt, at which time control is transferred to a special service routine. This routine first checks the contents of the location as an indicator: if the location contains 0000, then a null or idle character (code = 000) is transmitted; otherwise, control is transferred to the PDP-8 location specified by the contents of the location.

In this way a user function program may get control at interrupt-time by placing an appropriate address in the location.

When control comes to the user's service routine, 8-bit characters may then be transmitted by executing 'JMS* SNDCH' with the character code in bits 4-11 of the accumulator. The character is sent, and the return from the called subroutine occurs (with a cleared accumulator) the next time a transmitter flag interrupt occurs. Meanwhile, the main program flow may proceed independently.

Finally, when no more data transmission is to be done, the user's special service routine may set the contents of the location to 0000 and then return from the interrupt-time routine by executing 'JMP* INTRET'.

The following example will clarify the above information. In the example the following three character codes are transmitted: 226, 2ø1, 2ø3.

Main Program

```
        TAD CALPHA
        DCA* SNDCH          set pointer
        DCA SFLAG           clear indicator
           .
           .                arbitrarily long computation
           .
        TAD SFLAG           wait for transmission to complete
        SNA CLA
        JMP *-2
           .
           .                proceed
           .
CALPHA  PZE ALPHA
SFLAG   ***                 indicator
```

Special Service Routine

```
ALPHA   TAD =0226           send characters
        JMS* SNDCH
        TAD =02Ø1
        JMS* SNDCH
        TAD =02Ø3
        JMS* SNDCH
        DCA* SNDCH          stop sending
        ISZ SFLAG           set indicator
        JMP* INTRET         return
```

4.6.1.7.2 Receiving

The communication cell RCVCH contains a pointer to a location which is used as both an indicator and a subroutine entry point for receiving a character. Whenever the Dataphone receive flag is set (due to an incoming character), the DOGGIE interrupt-time service routine first checks the contents of the location as an indicator: if the location contains 0000, the character is ignored and the receive logic is cleared ("receive-active"). However, if the location is non-zero, control is transferred to the PDP-8 location specified by this

non-zero quantity, and bits 4-11 of the accumulator contain the character code received (bits 0-3 are ZERO).

Therefore, in order to receive, a user program must first place an appropriate address in the location. When the first character is received, the user program has control at interrupt-time. More characters may be received by executing 'JMS* RCVCH'. When this is done, the return from the called subroutine occurs the next time a character is received. Meanwhile, the main program flow may proceed independently.

Finally, when no more characters are to be received, the user's special service routine may set the contents of the location to 0000 and then return from the interrupt-time routine by executing 'JMP* INTRET'.

A special error condition is noted by DOGGIE: the receive-end flag of the Dataphone Interface is set whenever the receive logic loses the incoming signal. If this ever happens when a user program is waiting to receive a character, control is returned to the calling program with the accumulator set to all ONES.

The following example will clarify the above information. In the example, the first three characters received are saved and receiving is stopped.

Main Program

```
      TAD CBETA
      DCA* RCVCH             set pointer
      DCA RFLAG              clear indicator
         .
         .                   arbitrarily long computation
         .
      TAD RFLAG              wait until 3 chars have been received
      SNA CLA
      JMP *-2
         .
         .                   proceed
         .
CBETA PZE BETA
RFLAG ***                    indicator
```

Special Service Routine

```
BETA  DCA C1                 first
      JMS* RCVCH
      DCA C2                 second
      JMS* RCVCH
      DCA C3                 third
      DCA* RCVCH             stop receiving
      ISZ RFLAG              set indicator
      JMP* INTRET            return

C1    ***
C2    ***
C3    ***
```

4.6.2 Status Information

The interactive components of a user program obtain outputs from DOGGIE via the communication cells described below. All of the information represented by these cells is also available to interactive programs in the central computer except for the timer presented in Section 4.6.2.10.

4.6.2.1 Intensity

The communication cell INTENS contains the current overall picture intensity level in bits 9-11. Bits 0-8 of this word are ZERO's.

4.6.2.2 BLINK Mode and DIM Mode

The communication cell BLNDIM contains an indication which mode is currently prevailing, and if it is DIM mode, it indicates the current dimming intensity level. If BLINK mode prevails, BLNDIM contains 6301_8. If the prevailing mode is DIM mode, this cell contains 001n, where n is an octal digit whose value is the current dimming intensity level. Note that only the sign bit of this cell may be checked to determine which mode prevails.

4.6.2.3 Window Size

The communication cell WINSIZ contains the current window size in bits 10 and 11 as given below. Bits 0-9 of the word are ZERO's.

Bit 10	Bit 11	Window Size
0	0	FULL
0	1	HALF
1	0	FOURTH
1	1	EIGHTH

4.6.2.4 Window Position

The communication cell YWIND contains the y-coordinate of the position of the center of the window on the paper in bits 2-11. Bits 0 and 1 of this word are ZERO.

The communication cell XWIND contains the x-coordinate of the position of the center of the window on the paper in bits 2-11. Bits 0 and 1 of this word are ZERO.

4.6.2.5 Pseudo-Pen-Point Position

The communication cell YPSEUD contains a 12-bit pointer to the y-coordinate of the position of the pseudo-pen-point on the screen. The location where the position is stored contains the coordinate in bits 2-11. Bit 0 of the word is a ONE, and bit 1 is a ZERO.

The communication cell XPSEUD contains a 12-bit pointer to the x-coordinate of the position of the pseudo-pen-point on the screen. The location where the position is stored contains the coordinate in bits 2-11. Bit 0 of the word is a ONE, and bit 1 is a ZERO.

4.6.2.6 Tracking Indicator

The communication cell CHPSEU is made non-zero whenever there is a change in the location of the pseudo-pen-point due to tracking by the light pen.

After clearing this cell to 0000, a user program may then test its contents in order to determine when to perform a particular computation which depends upon the movement of the pseudo-pen-point. The cell CHPSEU may be cleared either by the 'START CURSOR' command (see Section 4.4.1.10) or directly by a user program.

4.6.2.7 Light Pen Handler

There are four communication cells dedicated to light pen hits: LPHIT1, LPHIT2, LPHIT3, LPHIT4.

When the contents of LPHIT1 is 0000, any light pen hit causes the Light Pen Handler to interpret the hit and output the following:

a) The contents of LPHIT1 is set to 3010_8 if the entity hit is a vertex, or it is set to 3000_8 if the entity hit is an arc.

b) The contents of LPHIT2 is set to the internal name of the entity hit.

c) If the entity hit is a vertex, the contents of LPHIT3 is set to the y-coordinate of the screen position of the hit vertex. If that position is off the screen, the nearest coordinate on the screen is recorded (either 0 or 1023).
If the entity hit is an arc, the contents of LPHIT3 is set to the y-coordinate of the screen position where the arc was hit.

d) If the entity hit is a vertex, the contents of LPHIT4 is set to the x-coordinate of the screen position of the hit vertex. If that position is off the screen, the nearest coordinate on the screen is recorded (either 0 or 1023).
If the entity hit is an arc, the contents of LPHIT3 is set to the x-coordinate of the screen position where the arc was hit.

Once a light pen hit occurs, another one is not allowed until the contents of LPHIT1 are set to 0000 either by the ALLHIT command or directly by a user program. Note that the ALLHIT command also clears

communication cell LPHIT2.

4.6.2.8 Created Internal Names

The communication cell VERCRN contains the current vertex created internal name. The communication cell ARCCRN contains the current arc created internal name.

4.6.2.9 Available Storage

The area of memory of the DEC-338 not assigned to DOGGIE or used for program overlays is maintained by a dynamic storage allocator as 19-word blocks. The communication cell FRBLKS contains the number of free blocks which are available for storage.

User programs may inspect this cell in order to determine whether there is enough available space for a particular operation to be performed. At present, it is a fatal error to run out of free storage. The following information can be used to estimate space requirements:

a) A vertex or arc with no label requires one storage block.

b) A vertex or arc with a label requires two storage blocks.

c) A storage block is used for approximately every eight entities (vertices or arcs) being displayed.

4.6.2.10 Timer

A user program may refer to two communication cells, HTIMER and LTIMER, which serve together as a 24-bit timer or real-time clock. The contents of LTIMER is incremented every 3 1/3 ms., and whenever an overflow occurs out of the high order bit of LTIMER, the contents of HTIMER is incremented.

Since DOGGIE does not make use of the contents of these cells, user programs may freely alter their contents. Since a program interrupt may occur at any time, it is suggested that LTIMER be set before HTIMER.

When the system is loaded, these cells both contain 0000.

4.6.2.11 Graph Status

Six communication cells are dedicated to output from the STATUS command: STAT1,STAT2,...,STAT6. When a STATUS command is given these cells are filled with all of the information about a particular entity. If no entity exists which satisfies the status request, the contents of STAT2 is set to 0000.

4.6.2.11.1 Vertex Status

If there is status output for a vertex, it is coded as follows:

STAT1

The twelve bits of this cell (0 through 11) are set as follows:

a) Bit 0 = ZERO if in DIM mode.
= ONE if in BLINK mode.

b) Bit 1 = ZERO if the vertex shape is not blinking.
= ONE if the vertex shape is blinking.

c) Bit 2 = ZERO

d) Bit 3 = ZERO if the label is not blinking.
= ONE if the label is blinking.

e) Bit 4 = ZERO if the vertex is not displayed.
= ONE if the vertex is displayed.

f) Bit 5 = ZERO if there is no label.
= ONE if there is a label.

g) Bit 6 = ZERO if a label is not displayed.
= ONE if a label is displayed.

h) Bit 7 = ZERO if light pen hits are disabled.
= ONE if light pen hits are enabled.

i) Bit 8 = ONE

j) Bits 9,10,11 = the vertex shape

STAT2

This cell contains the internal name of the vertex as a 12-bit integer.

STAT3

The two high order bits of this cell are ZERO and the low order ten bits contain the y-coordinate of the paper location of the vertex.

STAT4

The two high order bits of this cell are ZERO and the low order ten bits contain the x-coordinate of the paper location of the vertex.

STAT5

If the prevailing mode is BLINK mode, the contents of this cell has no relevance. If it is DIM mode, however, the twelve bits of this cell (0 through 11) are set as follows:

a) Bits 0,1,2 = ZERO

b) Bits 3,4,5 = dimming intensity level of the vertex shape.

c) Bits 6,7,8 = ZERO

d) Bits 9,10,11 = dimming intensity level of the label.

STAT6

This cell contains a 12-bit address which is one less than the position of the label buffer in the DEC-338 memory (field 0). If a label exists, as indicated by bit 5 of STAT1, this buffer contains from 3 to 16 words of information as follows:

First Word

This word contains the y-component of the offset as an 11-bit sign-magnitude integer, where the high order bit is ZERO and the next bit is the sign bit.

Second Word

This word contains the x-component of the offset as an 11-bit sign-magnitude integer, where the high order bit is ZERO and the next bit is the sign bit.

Remainder of the Buffer

The remainder of the buffer contains from 1 to 14 words with the text of the label. The unused buffer space is padded with 0000's. Text characters are packed two per word in the same format used in the specification of the label (see Section 4.5.3).

4.6.2.11.2 Arc Status

If there is status output for an arc, it is coded as follows:

STAT1

The twelve bits of this cell (0 through 11) are set as follows:

a) Bit 0 = ZERO if in DIM mode.
= ONE if in BLINK mode.

b) Bit 1 = ZERO if the arc itself is not blinking.
= ONE if the arc itself is blinking.

c) Bit 2 = ZERO if the arrow is not blinking.
= ONE if the arrow is blinking.

d) Bit 3 = ZERO if the label is not blinking.
= ONE if the label is blinking.

e) Bit 4 = ZERO if the arc is not displayed.
= ONE if the arc is displayed.

f) Bit 5 = ZERO if there is no label.
= ONE if there is a label.

g) Bit 6 = ZERO if a label is not displayed.
= ONE if a label is displayed.

h) Bit 7 = ZERO if light pen hits are disabled.

= ONE if light pen hits are enabled.

i) Bit 8 = ZERO

j) Bits 9,10 = 0,0 for no loop when C(STAT3)≠C(STAT4).

= 0,0 for East loop
= 0,1 for North loop
= 1,0 for West loop
= 1,1 for South loop
} when C(STAT3)=C(STAT4).

k) Bit 11 = ZERO if the arc does not have an arrow.

= ONE if the arc has an arrow.

STAT2

This cell contains the internal name of the arc as a 12-bit integer.

STAT3

The contents of this cell is the internal name of the vertex to which the arc is incident.

STAT4

The contents of this cell is the internal name of the vertex from which the arc is incident.

STAT5

If the prevailing mode is BLINK mode, the contents of this cell has no relevance. If it is DIM mode, however, the twelve bits of this cell (0 through 11) are set as follows:

a) Bits 0,1,2 = ZERO

b) Bits 3,4,5 = dimming intensity level of the arc itself.

c) Bits 6,7,8 = dimming intensity level of the arrow.

d) Bits 9,10,11 = dimming intensity level of the label.

STAT6

This cell contains a 12-bit address which is one less than the position of the label buffer in the DEC-338 memory (field 0). If a label exists, as indicated by bit 5 of STAT1, this buffer contains from 3 to 13 words of information as follows:

First Word

This word contains the y-component of the offset as an 11-bit sign-magnitude integer, where the high order bit is ZERO and the next bit is the sign bit.

Second Word

This word contains the x-component of the offset as an 11-bit sign-magnitude integer, where the high order bit is ZERO and the next bit is the sign bit.

Remainder of the Buffer

The remainder of the buffer contains from 1 to 11 words with the text of the label. The unused buffer space is padded with 0000's. Text characters are packed two per word in the same format used in the specification of the label (see Section 4.5.3).

4.6.3 DOGGIE Command Syntax

Section 4.4 described the DOGGIE Command Language in its pure form. When these commands are incorporated into interactive programs in the central computer the syntax used there is, for the most part, the pure form. However, user programs for the DEC-338, written under the PDPMAP Assembly System, must adhere to the syntax of MAP. A few macros are predefined for the user which allow for symbolic DOGGIE commands.

The fundamental macro is named "DOG". This macro is used to assemble one DOGGIE code word given as one argument. The argument may

be either a single DOGGIE word or any number of words or numbers separated by commas and enclosed within parentheses. Numbers are interpreted as decimal, and the DOG macro merely adds the values of the given terms and assembles one 12-bit word. Table 4-2, presented in Section 4.5, lists all defined DOGGIE words along with their values.

The role of the comma in pure DOGGIE Command Language separates 12-bit words, whereas separate lines must be used in user programs. The comma is used in a user program in place of a SPACE of pure DOGGIE Language. For example the DOGGIE command

START EXIST WHOLE VERTEX 2,CRNAME,PENPNT SCREEN,512

would be expressed in a user program as

```
DOG (START,EXIST,WHOLE,VERTEX,2)
DOG CRNAME
DOG (PENPNT,SCREEN)
DOG 512
```

Since MAP includes the OCT pseudo-operation, DOGGIE commands may include octal numbers. For example, the last line of the above four have been written as:

```
OCT 1000
```

Another useful macro is named "T" and is used for the assembly of text labels. This macro is defined to take one argument as a list of character terms separated by commas and enclosed within parentheses. Each given character term is encoded into 6-bit trimmed ASCII according to Table 4-3. Since macro parameters cannot include all special characters, special characters are represented by a pair of alphabetic characters. Table 4-4 indicates the correspondence between characters and character terms.

Table 4-4

Correspondence Between Characters and Character Terms

Text Character	Character Term	Text Character	Character Term
A-Z	A-Z	)	RP
[	LB	*	AS
\	BS	+	PL
]	RB	,	CM
↑	UA	-	MI
←	LA	.	DT
space	.	/	SL
!	EX	0-9	0-9
"	QU	:	CL
#	NM	;	SC
$	DL	<	LT
%	PC	=	EQ
&	AN	>	GT
'	AP	?	QM
(	LP		

As an example, the pure DOGGIE command

START EXIST LABEL VERTEX TEXT,1,@ HIT 'RETURN' WHEN DONE @

would be expressed in a user program as

```
DOG (START,EXIST,LABEL,VERTEX,TEXT)
DOG 1
T (H,I,T,.,AP,R,E,T,U,R,N,AP,.,W,H,E,N,.,D,O,N,E)
```

Note the T macro automatically terminates the sequence of character codes by a code of 00.

The DOG and T macros along with MAP pseudo-operations DEC, OCT, PZE, MZE, etc. are sufficient for assembling DOGGIE Command Language in user programs. In order for the assembled code to be interpreted by DOGGIE a call upon the interpreter must be made as:

```
JMS* DOGGIE
PZE Y-*
```

where Y is a location where DOGGIE commands have been assembled in sequential words until either a GOTO command is included or a command of 0000 is given which terminates the list of commands.

As an aid to the programmer, two macros are predefined to surround a list of commands placed in line with PDP-8 code. A call upon the DOGGIE macro causes a pair of words to be assembled which constitutes a call upon the DOGGIE interpreter, with the relative pointer indicating where the command list begins. The command list is assembled at a different place in memory according to the "GRAPHS" Location Counter, and the DOGGIE macro places that Location Counter in control. Therefore, a list of DOGGIE commands is meant to follow the call upon the DOGGIE macro. This list is terminated by a call upon the ENDDOG macro which first assembles the termination command of 0000 and then switches back

to the normal Blank Location Counter. The GRAPHS Location Counter is begun immediately following the end of the assembled program.

This feature makes programming easier, and especially makes coding readable, as can be observed in the example presented in the next section.

4.6.4 Sample User Program

This section presents a rather small, yet complete, example of a user program which is written for the PDPMAP Assembler. The program begins at location 6000_8 in the DEC-338, which is rather standard since this is the beginning of the area used for program segments. It initially places two messages (for the user) at the lower left corner of the screen by using labels of vertices whose shape is null. As the messages indicate, the program changes the shape of all vertices seen by the light pen to be square (vertex shape 6) until either pushbutton 11 or the manual interrupt button is depressed, at which time the Graph Monitor is restarted. Note since the Monitor will define new vertices with internal names the same as those used for messages, it is not necessary to delete them before calling the Graph Monitor.

A source listing of the sample user program follows on the next page.

4.7 Interactive Programs in the IBM 7040

Section 4.6 has described the interactive components of DOGGIE which supplement the DOGGIE Command Language to support interactive user programs. The Interactive Graph Theory System, as seen by a user, includes complete graph manipulation tools which are user programs written by the author. The facility for other user programs to be added is included in the design, but it is expected that the system will grow

```
          OCTORG 6000
          JMS* MANINT       CLEAR FLAG
          NOP
          JMS* PBCLR        CLEAR PB 11
          OCT 7776
          DUGGIE
          DOG (START,LTPEN,WHOLE,VERTEX)
          DOG ALL
          DOG (START,EXIST,WHOLE,VERTEX,LIST)
          OCT 7777,2024,2024
          OCT 7776,2054,2024
          OCT 0             END OF LIST
          DOG (START,EXIST,LABEL,VERTEX,TEXT,LIST)
          OCT 7776
          T (P,O,I,N,T,.,T,O,.,V,E,R,T,I,C,E,S,.,T,O,.,B,E)
          OCT 7777
          T (M,A,D,E,.,S,Q,U,A,R,E,CM,.,P,B,.,1,1,.,T,O,.,S,T,O,P)
          OCT 0             END OF LIST
          DOG (START,DSPLAY,WHOLE,VERTEX,LIST)
          OCT 7776,7777,0
          ENDDOG
     MORE DCA LPHIT1        ALLOW HITS
     WAIT JMS* PBSKIP       CHECK PB 11
          OCT 0001
          JMS* MANINT       CHECK 'INTERRUPT'
          JMP* RESTRT       RESTART GRAPH MONITOR
          TAD LPHIT1
          SNA CLA           CHECK FOR HIT
          JMP WAIT          WAIT FOR SOMETHING
          TAD LPHIT2        HIT OCCURED
          DCA NOWSQ         SET UP NAME
          DUGGIE
          DOG (START,EXIST,SHAPE,VERTEX,6)
    NOWSQ ***
          ENDDOG
          JMP MORE
          END
```

more readily by writing interactive ALLA programs for the IBM 7040. Such programs are easier to write, easier to read, and particularly significant is the graph-theoretical and arithmetic support which the ALLA environment provides.

When the user at the terminal calls upon the central computer to execute a program already in the system, a particular user program is given control in the DEC-338 which makes the terminal a slave to the central computer. As such, it is capable of receiving over the Dataphone line nearly all possible DOGGIE commands. In addition, this dedicated use of the DEC-338 allows for some information flow from the terminal to the central computer in response to a request from the latter. For the most part, this is in the form of directly copying various communication cells containing status information.

The remainder of this section describes from the programmer's viewpoint the facilities of the Interactive Graph Theory System which extend ALLA to interface with DOGGIE.

4.7.1 DOGGIE Commands

The DOGGIE Command Language may be included in ALLA programs in order to control the existence and display of graphs at the display terminal. All commands described in Sect. 4.4 may be used except LOAD, GOTO, or LOADGO. This restriction does not outlaw the calling of user programs into operation from the central computer; another method is available for this operation as described in Section 4.7.6.2.

Unlike user programs, ALLA programs do not include explicit calls upon the DOGGIE interpreter. Instead there are three types of statements which are used to output one or more 12-bit words for DOGGIE. This outputting is buffered in the memory of the IBM 7040, on the disk of

the 7040, and also in the PDP-8 and DEC-338. Therefore, statements which output to DOGGIE may be scattered among other types of statements. This gives the ALLA programmer an easier means of expressing DOGGIE operations than the programmer of user programs.

When the DOGGIE interpreter is called directly by a user program, it is treated as a subroutine (see Section 4.6.1.1). The DOGGIE command of 0000 is then interpreted as a terminator of a sequence of commands and results in return of control from the interpreter back to the calling program. Since this operation has no parallel during the use of DOGGIE within ALLA programs, the meaning of the DOGGIE command of 0000 is lost. Therefore, the convention has been established to treat this special case as a no-operation command.

The DOG statement is the fundamental statement type for the outputting of DOGGIE commands. It is used to indicate what follows on the same card image is to be coded into 12-bit words and placed into the buffer for DOGGIE commands. In general, a complete symbolic DOGGIE command appears on one card, as shown in the examples:

```
DOG BLINKM
DOG SETWIN HALF
DOG START BLINK ARROW ARC,ALL
```

However, separate statements may be used to specify individual 12-bit words. For example, the previous DOGGIE command could be given as:

```
DOG START BLINK ARROW ARC
DOG ALL
```

The DOG statement may include DOGGIE words along with decimal numbers in the syntax of DOGGIE commands presented in Section 4.4. In addition, the terms of a DOG statement may be ALLA integer expressions. Each expression of a DOG statement whose value is negative is evaluated as

the 12-bit two's complement quantity.

The programmer may use symbolic DOGGIE words up to the <u>term</u> in a DOG statement where he employs one of the following special characters:

+ - * (

The remainder of the DOG statement is then treated as any number of integer expressions. Using this facility, the arrows of the first seven arcs (internal names 1 through 7) of a graph may be blinked by:

```
          DO 10 I = 1,7
   10     DOG START BLINK ARROW ARC,(I)
```

Note that the "(" is the indication that what follows is not a symbolic DOGGIE word, but is instead an integer expression. The following sequence of statements has the same effect as the above two statements, but outputs fewer DOGGIE words:

```
          DOG START BLINK ARROW ARC LIST
          DO 10 I = 1,7
   10     DOG (I)
          DOG 0
```

The above example is not particularly useful, but its idea can be extended to a sequence which blinks the arrows of those arcs belonging to set ARCS. In the following example, the property name INNAME is used as an integer property whose value is the internal name of the entity with which it is associated.

```
          DOG START BLINK ARROW ARC LIST
          THROUGH 10 FORALL A IN ARCS
   10     DOG (INNAME(A))
          DOG 0
```

The use of integer expressions, or particularly integer variables, is a significant addition to DOGGIE commands. The positioning of a vertex at some paper or screen location may be done by a DOGGIE command

with integer expressions representing computations for the y- and x-coordinates. Further uses of this feature will become apparent when communication cells are introduced.

The one aspect of pure DOGGIE command language which is not available with the DOG statement is the specification of text characters for a label. The DOGSTRING statement is available for encoding labels which are completely known at the time the program is compiled. These are often used for instructional messages to the user. The DOGSTRING statement includes the label preceded by and followed by any character which does not appear within the label itself. For example, the apostrophe is used as the signal character in the following DOGGIE command:

```
DOG START EXIST LABEL VERTEX TEXT,4093
DOGSTRING 'POINT TO VERTEX TO BE MOVED'
```

The third type of statement available for outputting DOGGIE commands is DOGTEXT, which is used to construct labels at execution time. A DOGTEXT statement includes one or more integer expressions, each separated by a comma. Each given expression is evaluated as a 36-bit word and treated as six 6-bit trimmed ASCII character codes. The first 6-bit code of 00 detected causes the remainder of the given arguments to be ignored. The following is an example of the use of DOGTEXT, where the function ITOA translates six character codes of one 36-bit word from IBM code to trimmed ASCII.

```
DOG START EXIST LABEL VERTEX TEXT,4095
DOGTEXT LABEL(1),LABEL(2),ITOA(6HIS NOT),ITOA(6H NEAR.)
```

4.7.2 Extending the DOGGIE Language

Since ALLA includes the facility for writing subroutines, commonly used groups of DOGGIE commands may be replaced by appropriate subroutine

calls upon a subroutine which has the same effect. Two examples of this form of extending the language are given.

4.7.2.1 User Messages

A subroutine named MESSAG has been made a standard part of the available facilities in order to aid the programmer in giving a message to the user. The subroutine has one argument which is an integer quantity specifying the line number of the message which follows the call. Lines are numbered 1,2,3,..., beginning at the lower left corner of the screen and going up the left edge. The following source listing of this subroutine should be sufficiently clear to the reader to make its characteristics obvious.

```
SUBROUTINE MESSAG(I)
INTEGER I,J
J = -I
DOG START EXIST WHOLE VERTEX,(J)
DOG SCREEN+24*I-4
DOG SCREEN+20
DOG START EXIST LABEL VERTEX TEXT,(J),0
DOG START DSPLAY WHOLE VERTEX,(J)
DOG START EXIST LABEL VERTEX TEXT,(J)
RETURN
END
```

The call upon the MESSAG subroutine must be followed by a DOGSTRING or DOGTEXT statement to specify the message. An example of such a call follows:

```
CALL MESSAG(3)
DOGSTRING *POINT TO FROM-VERTEX*
```

4.7.2.2 Clearing and Setting Pushbuttons

Another reason for extending the DOGGIE language is to overcome its inadequacies. The command to clear selected pushbuttons according

to bit pattern is convenient for use in PDPMAP user programs, but rather inappropriate within ALLA programs. The following subroutine may be used to clear one selected pushbutton according to the given integer.

```
SUBROUTINE CLRPB(I)
INTEGER I
DOG CLRPB,-1-2**(11-I)
RETURN
END
```

An equivalent version of the above subroutine and a similar one named SETPB for setting pushbuttons are included as standard facilities in the Interactive Graph Theory System. Since they were coded in assembly language, they were written to accept any number of arguments as integer expressions whose values range between 0 and 11. For example, the following two subroutine calls clear pushbuttons 7, 8, 9, 10 and set pushbuttons 4 and 5:

```
CALL CLRPB(7,8,9,10)
CALL SETPB(5,4)
```

4.7.3 Values of DOGGIE Words

Since an integer expression may be used to specify an expression of a DOG statement, the programmer may wish to set the value of an integer variable equal to the value of a DOGGIE expression. The DOGSET statement is available for this purpose. This type of statement includes an integer assignment statement, except the expression to the right of the equal sign is interpreted as a DOGGIE expression. The following examples are included to clarify the syntax of this type of statement:

```
DOGSET SELVT = START EXIST LABEL VERTEX TEXT
DOGSET LOOP(1) = LOOPE
DOGSET LOOP(2) = LOOPN
```

Note the first example is a case where a programmer may wish to abbre-

viate a frequently used DOGGIE expression. The second and third lines indicate another use of the DOGSET statement. The idea behind them is that a computation yielding an integer between 1 and 4 could be directly encoded into a DOGGIE command to affect a loop orientation. This, of course, presumes LOOP(3) and LOOP(4) have been similarly assigned values.

4.7.4 Status of the DEC-338

During execution of an ALLA program in the Interactive Graph Theory System, the program may request standard status information from the DEC-338. This status is the means by which the user inputs to the system via Teletype, light pen, and pushbuttons. Most of this information is a copy of various communication cells of DOGGIE which were described in Section 4.6.2. When the status information enters the IBM 7040, it is distributed to a set of communication cells which are integer and logical variables which all ALLA program decks may reference. This section describes the communication cells which the programmer may reference. There is no need for him to declare any of the communication cell variables since their allocation is automatically handled in each ALLA subprogram deck. It should be remembered that their contents are changed only when there is a request for status information. The various ways in which status requests occur will be presented in Section 4.7.6.

Since the size of the memory word is 36 bits in the IBM 7040, each integer variable is assigned that many bits in order to represent an integer. Many items of status information occupy only 12 bits since they originated from communication cells in the DEC-338. These 12-bit quantities are kept in the low-order end of the 36-bit word, i.e. bits 24-35. Where bit numbers must be given in the following subsections they are based upon the 36-bit word.

4.7.4.1 Intensity

The communication cell INTENS is an integer variable whose value is between 0 and 7 which indicates the overall picture intensity level.

4.7.4.2 BLINK Mode and DIM Mode

The communication cell BLNDIM is an integer variable which indicates which mode prevails, and if it is DIM mode, it indicates the current intensity level. If BLINK mode prevails, the value of BLNDIM is -1. If the prevailing mode is DIM mode, the value of BLNDIM is between 0 and 7 which indicates the dimming intensity level.

4.7.4.3 Window Size

The communication cell WINSIZ is an integer variable whose value is between 0 and 3 which indicates the size of the window as given below.

Value of WINSIZ	Window Size
0	FULL
1	HALF
2	FOURTH
3	EIGHTH

4.7.4.4 Window Position

The communication cell YWIND is an integer variable whose value is the y-coordinate of the position of the center of the window on the paper. This is an integer between 0 and 1023.

The communication cell XWIND is an integer variable whose value is the x-coordinate of the position of the center of the window on the paper. This is an integer between 0 and 1023.

4.7.4.5 Pseudo-Pen-Point Position

The communication cell YPSEUD is an integer variable whose value is the y-coordinate of the position of the pseudo-pen-point on the screen.

This is an integer between 0 and 1023.

The communication cell XPSEUD is an integer variable whose value is the x-coordinate of the position of the pseudo-pen-point on the screen. This is an integer between 0 and 1023.

4.7.4.6 Tracking Indicator

The communication cell CHPSEU is an integer variable whose value is cleared to 0 by the 'START CURSOR' command (or directly by a user program within the DEC-338). The value of CHPSEU is 1 after there is a change in the location of the pseudo-pen-point due to tracking by the light pen.

4.7.4.7 Light Pen Handler

There are four communication cells which are integer variables dedicated to light pen hits: LPHIT1, LPHIT2, LPHIT3, LPHIT4. Unless altered by the ALLHIT command or by a user program in the DEC-338, their contents reflect the last light pen hit which was interpreted. In order for the Light Pen Handler to interpret a hit, the communication cell LPHIT1 (in the DEC-338) must be clear. It may be cleared by the ALLHIT command or directly by a user program in the DEC-338. Note that the ALLHIT command also clears communication cell LPHIT2.

When the Light Pen Handler interprets a hit, it outputs the following:

a) The value of LPHIT1 is set to 3010_8 if the entity hit is a vertex, or it is set to 3000_8 if the entity hit is an arc.

b) The value of LPHIT2 is set to the internal name of the entity hit. This is an integer between 1 and 4095.

c) If the entity hit is a vertex, the value of LPHIT3 is set to the y-coordinate of the screen position of the hit vertex. If that position is off the screen, the nearest

coordinate on the screen is recorded (either 0 or 1023). This is an integer between 0 and 1023.

If the entity hit is an arc, the value of LPHIT3 is set to the y-coordinate of the screen position where the arc was hit. This is an integer between 0 and 1023.

d) If the entity hit is a vertex, the value of LPHIT4 is set to the x-coordinate of the screen position of the hit vertex. If that position is off the screen, the nearest coordinate on the screen is recorded (either 0 or 1023). This is an integer between 0 and 1023.

If the entity hit is an arc, the value of LPHIT3 is set to the x-coordinate of the screen position where the arc was hit. This is an integer between 0 and 1023.

4.7.4.8 Created Internal Names

The communication cell VERCRN is an integer variable whose value is the vertex created internal name. The communication cell ARCCRN is an integer variable whose value is the arc created internal name. Each of these may be integers between 1 and 4095.

4.7.4.9 Available Storage

The area of memory of the DEC-338 not assigned to DOGGIE or used for program overlays is maintained by a dynamic storage allocator as 19-word blocks. The communication cell FRBLKS is an integer variable whose value is the number of free blocks available in the DEC-338.

A program may reference this cell in order to determine whether there is enough available space for a particular operation to be performed. At present, it is a fatal error to run out of free storage. The following information can be used to estimate space requirements:

a) A vertex or arc with no label requires one storage block.

b) A vertex or arc with a label requires two storage blocks.

c) A storage block is used for approximately every eight entities (vertices or arcs) being displayed.

4.7.4.10 Pushbuttons

There are two methods which ALLA programs may use to determine the settings of pushbuttons. Since pushbuttons 0 and 6 are used internally by DOGGIE they always appear as being cleared according to the status information. The communication cell PBS is an integer variable whose value is a bit pattern corresponding to the 12 pushbuttons. Bit 24 corresponds to pushbutton 0, bit 25 corresponds to pushbutton 1, ..., and bit 35 corresponds to pushbutton 11. A pushbutton which is clear is represented by its corresponding bit set to ZERO, and a set pushbutton is represented by its corresponding bit set to ONE.

It may be convenient to reference the cell PBS in order to check for a pattern match of some of the pushbuttons, but it is more common to check only one pushbutton at a time. For this purpose a logical function PB is available to the ALLA programmer. This function is used with one integer argument whose value should be between 0 and 11 in order to specify a particular pushbutton. The value of the function is .TRUE. if and only if the selected pushbutton is set (according to the communication cell PBS).

4.7.4.11 Graph Status

Twelve communication cells in the IBM 7040 are dedicated to output from the STATUS command. Seven of these cells are integer variables: STAT1, STAT2, STAT3, STAT4, STAT5, YOFF, XOFF. The remaining five cells are members of an integer list (one-dimensional array): LABEL(5).

Unless altered by a user program in the DEC-338, the values of these cells reflect the results of the last STATUS command. When a STATUS command is given these cells are filled with all of the information about a particular entity. If no entity exists which satisfies the status request, the value of STAT2 is set to 0.

4.7.4.11.1 Vertex Status

If there is status output for a vertex, it is coded as follows:

STAT1

The low-order 12 bits of STAT1 (24 through 35) are set as follows:

a) Bit 24 = ZERO if in DIM mode.
   = ONE if in BLINK mode.

b) Bit 25 = ZERO if the vertex shape is not blinking.
   = ONE if the vertex shape is blinking.

c) Bit 26 = ZERO

d) Bit 27 = ZERO if the label is not blinking.
   = ONE if the label is blinking.

e) Bit 28 = ZERO if the vertex is not displayed.
   = ONE if the vertex is displayed.

f) Bit 29 = ZERO if there is no label.
   = ONE if there is a label.

g) Bit 30 = ZERO if a label is not displayed.
   = ONE if a label is displayed.

h) Bit 31 = ZERO if light pen hits are disabled.
   = ONE if light pen hits are enabled.

i) Bit 32 = ONE

j) Bits 33,34,35 = the vertex shape

STAT2

The value of STAT2 is the internal name of the vertex. It is an integer between 1 and 4095.

STAT3

The value of STAT3 is the y-coordinate of the paper location of the vertex. It is an integer between 0 and 1023.

STAT4

The value of STAT4 is the x-coordinate of the paper location of the vertex. It is an integer between 0 and 1023.

STAT5

If the prevailing mode is BLINK mode, the value of STAT5 has no relevance. If it is DIM mode, however, the low-order nine bits of STAT5 (27 through 35) are set as follows:

a) Bits 27,28,29 = dimming intensity level of the vertex shape.

b) Bits 30,31,32 = ZERO

c) Bits 33,34,35 = dimming intensity level of the label.

YOFF

If the vertex has a label, as indicated by bit 29 of STAT1, the value of YOFF is the y-component of the offset contained in the low-order 11 bits (25 through 35). This quantity is coded as a sign-magnitude integer where bit 25 is the sign bit.

XOFF

If the vertex has a label, as indicated by bit 29 of STAT1, the value of XOFF is the x-component of the offset contained in the low-order 11 bits (25 through 35). This quantity is coded as a sign-magnitude integer where bit 25 is the sign bit.

LABEL(1),...,LABEL(5)

If the vertex has a label, as indicated by bit 29 of STAT1, these cells are packed with 6-bit trimmed ASCII codes of the label. The codes are packed six per word and unused positions are padded with 00's. Note a vertex label may have at most 27 characters.

4.7.4.11.2 Arc Status

If there is status output for an arc, it is coded as follows:

STAT1

The low-order 12 bits of STAT1 (24 through 35) are set as follows:

a) Bit 24 = ZERO if in DIM mode.
= ONE if in BLINK mode.

b) Bit 25 = ZERO if the arc itself is not blinking.
= ONE if the arc itself is blinking.

c) Bit 26 = ZERO if the arrow is not blinking.
= ONE if the arrow is blinking.

d) Bit 27 = ZERO if the label is not blinking.
= ONE if the label is blinking.

e) Bit 28 = ZERO if the arc is not displayed.
= ONE if the arc is displayed.

f) Bit 29 = ZERO if there is no label.
= ONE if there is a label.

g) Bit 30 = ZERO if a label is not displayed.
= ONE if a label is displayed.

h) Bit 31 = ZERO if light pen hits are disabled.
= ONE if light pen hits are enabled.

i) Bit 32 = ZERO

j) Bits 33,34 = 0,0 for no loop when $C(STAT3) \neq C(STAT4)$.

= 0,0 for East loop
= 0,1 for North loop
= 1,0 for West loop
= 1,1 for South loop
when $C(STAT3)=C(STAT4)$.

k) Bit 35 = ZERO if the arc does not have an arrow.
= ONE if the arc has an arrow.

STAT2

The value of STAT2 is the internal name of the arc. It is an integer between 1 and 4095.

STAT3

The value of STAT3 is the internal name of the vertex to which the arc is incident. It is an integer between 1 and 4095.

STAT4

The value of STAT4 is the internal name of the vertex from which the arc is incident. It is an integer between 1 and 4095.

STAT5

If the prevailing mode is BLINK mode, the value of STAT5 has no relevance. If it is DIM mode, however, the low-order nine bits of STAT5 (27 through 35) are set as follows:

a) Bits 27,28,29 = dimming intensity level of the arc itself.

b) Bits 30,31,32 = dimming intensity level of the arrow.

c) Bits 33,34,35 = dimming intensity level of the label.

YOFF

If the arc has a label, as indicated by bit 29 of STAT1, the value of YOFF is the y-component of the offset contained in the low-order 11 bits (25 through 35). This quantity is coded as a sign-magnitude integer

where bit 25 is the sign bit.

XOFF

If the arc has a label, as indicated by bit 29 of STAT1, the value of XOFF is the x-component of the offset contained in the low-order 11 bits (25 through 35). This quantity is coded as a sign-magnitude integer where bit 25 is the sign bit.

LABEL(1),...,LABEL(5)

If the arc has a label, as indicated by bit 29 of STAT1, these cells are packed with 6-bit trimmed ASCII codes of the label. The codes are packed six per word and unused positions are padded with 00's. Note an arc label may have at most 23 characters.

4.7.4.12 Teletype Input

When the DEC-338 is acting as a slave to the IBM 7040 during execution of an interactive ALLA program, the dedicated user program in the DEC-338 echos and buffers Teletype input of up to one 64-character line at a time. Whenever the user types 'RETURN', the line which it terminated is ready to be sent to the IBM 7040. The line is sent along with status information on the next time there is a request from the 7040 for DEC-338 status. When this occurs a LINE FEED is typed out so the user may proceed typing another input line if he so desires. Any characters he may attempt to type before the LINE FEED is given are lost and are not echoed.

The ALLA programmer may test communication cell TTYIN which is a logical variable whose value indicates whether the last status sent included a line of Teletype input. The value of TTYIN is .TRUE. if Teletype input is available.

Eleven communication cells, members of an integer list (one-dimensional array) named TTYBUF, constitute the Teletype input buffer. This buffer has relevant contents only when the value of TTYIN is .TRUE. This input buffer is packed six characters per word as trimmed ASCII code, with a characteristic code of 00 following the last input character (unless the line is full).

4.7.5 Manual Interrupt Button

The convention has been established (see Section 4.6.1.3) which assigns a meaning of termination to the manual interrupt button. Interactive programs in the IBM 7040 cannot test for the condition that the button has been pushed. Instead, the convention has dictated a more automatic response. Pushing this button during interactive execution causes that execution to terminate, and an indicative message is shown on the screen.

4.7.6 Requests for DEC-338 Status and Interaction

The communication cells of the IBM 7040 described in Section 4.7.4 are filled with status information from the DEC-338 only when the program in the IBM 7040 requests this status. There are three additional types of statements plus two subroutines in the ALLA environment which the programmer may call upon. These statements and subroutine calls cause the generation of certain DOGGIE GOTO commands in order to get the dedicated user program running in the DEC-338 to perform an alternate task. The programmer, therefore, should not have uncompleted DOGGIE commands pending at the points in his program where these requests are made. In order to avoid such problems any of this class of interactive command first outputs three words of 0000 for DOGGIE. This is followed by the GOTO command. In order to insure that all of the DOGGIE commands

will reach the DEC-338, a flush of the DOGGIE command buffer is also performed. The programmer is also given the power to flush this buffer as described in Section 4.7.6.3

4.7.6.1 Executable Statements: GETSTATUS, WAITCHANGE, ESCAPE

The ALLA programmer may include the following executable statement within a program:

GETSTATUS

At execution time, this statement causes the DEC-338 status information to fill the communication cells. Control does not pass through this statement until the information transfer is complete. An automatic GETSTATUS operation is always performed at initialization time, directly after loading, of every application of the ALLA system.

The dedicated user program which controls the DEC-338 during the execution of interactive IBM 7040 programs keeps track of the status information it sends to the IBM 7040 concerning pushbutton status, light pen hit status, and Teletype input status. Since interactive programs often require the user to perform some input action in order to proceed, another executable statement may be included within an ALLA program:

WAITCHANGE

At execution time, this statement causes the DEC-338 to send status information only after a change in the status of the pushbuttons, Light Pen Handler, or Teletype input (a complete input line). Control does not pass through this statement until a status information transfer occurs and is complete.

The third additional type of statement which results in DEC-338 status information filling the IBM 7040 communication cells is used more sparingly, for it releases control of the DEC-338 to the user for local use with the "understanding" that the user will later return to the interactive execution mode by selecting the "Resume Execution" option available under the Graph Monitor (described in Section 5.7). The form of the ALLA statement is:

ESCAPE

At execution time, this statement causes the DEC-338 to escape control of the IBM 7040 and the Graph Monitor is made available to the user. A typical use of this feature is during an interactive program when the user must make significant modifications to the graph he is currently handling. The local features of manipulation available through the Graph Monitor can be used. Meanwhile, the IBM 7040 hangs waiting for the DEC-338 status information, which will finally be sent when the user "resumes execution." Control does not pass the ESCAPE statement until a status information transfer occurs and is complete.

If the user decides to terminate interactive execution at a time when he is using the Graph Monitor as a result of an ESCAPE, he must first resume execution and then use the manual interrupt button to terminate.

4.7.5.2 Subroutines

A very useful feature of the Interactive Graph Theory System is the facility for interactive programs running in the central computer to call upon specially written user programs which can provide interactive operations many times faster than the interactions between the central computer and the display terminal. These programs are written

in the same manner as any local user program using the PDPMAP Assembly System, except there is some restriction on their placement in the DEC-338 memory, and they must terminate in particular ways. Also, the conventional use of the manual interrupt button must be handled. Appendix 2 includes the rules governing the writing of these user programs. There is a system capacity for nine such special user programs to be on the disk of the DEC-338 at any one time. The file names used for the programs must be USE1,USE2,...,USE9.

4.7.6.2.1 USER

A user program of the type described above is called into action by the following subroutine call in an interactive ALLA program:

CALL USER(n)

where n is an integer expression whose value is between 1 and 9. According to the argument given, the corresponding user program is started in the DEC-338. That program must be written so that upon termination it will send status information to the IBM 7040 and resume the dedicated user program which makes the DEC-338 a slave to the central computer. Meanwhile, after the subroutine call to USER has been made, the IBM 7040 hangs waiting for DEC-338 status information. Control does not pass through this statement until a status information transfer occurs and is complete.

A user program of the type being considered here may be designed to perform only a particular phase of an interactive program which is complete in itself. However, a user program may require the communication of information back to the central computer as a result of its operating. The means by which such communication may occur is via those communication cells which are copied when DEC-338 status information is sent out.

This type of communication is used by the SELECT routine given as a complete example in Appendix 2 and described in the following subsection.

4.7.6.2.2 SELECT

Since the interactive operation of having the user select a vertex or arc by pointing is so common in using the Interactive Graph Theory System, the author has provided one user program which can be called into action from the IBM 7040 to perform this function. The program is named SELECT, but it is saved on the DEC-338 disk with the name USE9. Therefore, the following subroutine call may be used to activate the program:

```
CALL USER(9)
```

Since this particular program is considered a standard feature of the Interactive Graph Theory System, a special subroutine call, equivalent to the above call, may be included in ALLA programs:

```
CALL SELECT
```

The characteristics of this program are presented here for the ALLA programmer who wishes to make use of it. The PDPMAP assembly listing of the program is included in Appendix 2 as an example of a user program. A simple example of the use of the SELECT routine follows below.

4.7.6.2.2.1 Characteristics

When the SELECT routine is called, it assumes the programmer has placed a message on the screen at message lines 3, 4, etc. which indicate to which entities the user should point. For example, the following statements might be used:

```
CALL MESSAG(3)
DOGSTRING 'POINT TO A STARTING VERTEX'
```

The SELECT routine always begins by placing the following message at message line 2:

OR PB 10 FOR NO SELECTION

The ALLA program must also enable the light pen status of those entities which may be selected. For example, if the user is to select any vertex, the following two statements might be used:

DOG STOP LTPEN WHOLE ARC,ALL
DOG START LTPEN WHOLE VERTEX,ALL

The routine then sets the prevailing mode to BLINK mode, clears pushbuttons 10 and 11, and clears the communication cell LPHIT1 so that light pen hits may occur. When a hit does occur on a vertex or arc, that entity is made to blink, and if then another entity is hit, the previously hit one stops blinking, and the freshly hit one blinks. When the first hit occurs, message lines 1 and 2 are changed to:

PB 11 SELECTS BLINKING ONE

OR POINT TO ANOTHER ONE

These lines do make sense to the user in the context of message lines 3, 4, etc., which remain displayed during the pointing.

If pushbutton 10 is ever pushed (even after selection has occurred), any blinking is stopped, the contents of communication cell LPHIT1 is set to 1, LPHIT2 is set to 0, and normal termination occurs. If pushbutton 11 is pushed after a selection has been made, the contents of the four communication cells associated with the Light Pen Handler are set to indicate the light pen hit which caused the blinking entity to blink, and normal termination occurs. The selected blinking entity remains blinking.

Normal termination of the SELECT routine consists of eliminating the messages on message lines 1, 2, 3, and 4. Finally, DEC-338 status is returned to the IBM 7040.

During the execution of SELECT, a line of Teletype input may be prepared, but it does not affect the operation of the user program. In keeping with the established conventions, the SELECT routine monitors the manual interrupt button, and it interprets the hitting of this button as a user command to terminate interactive execution.

4.7.6.2.2.2 Example

The following interactive ALLA subprogram is an example of the use of the SELECT routine. The program requests the user to select any vertex. Upon selection, the program dims down the graph to intensity level 3, and brightens all vertices of the graph which are reachable from the chosen vertex by travelling along any arc which emanates from that vertex. The practical value of the function is negligible, but it demonstrates the use of SELECT. The one argument to the routine is an entity which is the graph in the ALLA structure corresponding to the one being displayed at the DEC-338. The property INNAME is an integer property of each vertex and arc which is the internal name of that entity. The following is a source listing of the interactive ALLA subroutine NXTNEI:

```
      SUBROUTINE NXTNEI(G)
      ENTITY G,A
      INTEGER INNAME
      DOG STOP LTPEN WHOLE ARC, ALL
      DOG START LTPEN WHOLE VERTEX, ALL
      CALL MESSAG(4)
      DOGSTRING 'TO SHOW NEXT NEIGHBORS'
      CALL MESSAG(3)
      DOGSTRING ' POINT TO ANY VERTEX'
      CALL SELECT
      DOG DIMM 3
      IF (LPHIT2 .EQ. 0) RETURN
      CALL MESSAG(1)
      DOGSTRING 'NEXT NEIGHBORS ARE SHOWN'
      DOG START DIM WHOLE VERTEX 7 LIST, 4095
      THROUGH 100 FORALL A IN RELM(G)
      IF (INNAME(LELM(A)) .NE. LPHIT2) GOTO 100
      DOG (INNAME(RELM(A)))
100   CONTINUE
      DOG 0
      RETURN
      END
```

4.7.6.3 Buffering of DOGGIE Commands

In Section 4.7.1, the use of a buffer for DOGGIE commands was introduced. There are in fact many places where these words are buffered between the execution of a DOG statement and the final interpretation in the DEC-338. The programmer, as one would hope, may normally ignore the detailed underlying workings of the system he is using. The one occasional exception to this philosophy arises since command words are buffered, at the first level, in the memory of the IBM 7040. When commands are yet in this particular buffer, they cannot get to the DEC-338. This DOGGIE command buffer must either be completely filled with 320 12-bit DOGGIE words or the buffer must be flushed.

As explained in the beginning of Section 4.7.6, all interactive statements which cause special action to be performed by the DEC-338 create a GOTO command followed by a buffer flush. Therefore, at any

initial point in a program when input from the user is requested, the buffer has been flushed and thus all DOG commands executed have been interpreted by DOGGIE.

Although it may be unnecessary computationally, it may be an aesthetic requirement for an interactive program to flush the DOGGIE command buffer explicitly. This need might arise when using the Interactive Graph Theory System to process an algorithm which involves much computation. In such a case, very few DOGGIE commands might be generated as indicative monitoring of the algorithm, perhaps, for example, only every few seconds. In order for the user to follow the monitoring, the program must flush the buffer whenever necessary. The following statement is used in an interactive ALLA program to flush the DOGGIE command buffer:

DOGFLUSH

An attempt to flush the DOGGIE command buffer when it has already been flushed and is still empty results in no output operation. One restriction of which the programmer must beware is a buffer flush must not be done when there are any incompleted DOGGIE commands pending.

4.7.7 Input of Graphs

There are two distinct ways in which an interactive ALLA program may read in graphs. The more common source of a graph is the graph being displayed at the DEC-338. The following statement is used in an interactive ALLA program to read into the ALLA structure the graph being displayed at the terminal:

GETGRAPH ent

where ent is a nonsubscripted entity. This statement causes a special

DOGGIE GOTO command to be sent to the DEC-338 in the same way in which other interactive statements operate (see Section 4.7.6). However, DEC-338 status is not updated in the IBM 7040 as a result of executing this statement. This statement must not be used at a point in the program when there are any uncompleted DOGGIE commands pending. When the statement is executed, the IBM 7040 hangs waiting for the coded graph to be sent from the DEC-338. Control does not pass the GETGRAPH statement until the entire displayed graph is sent and has been set up in the ALLA structure as described below.

The other source of graphs is the MULTILIST Data File. When the user initiates execution through the Graph Monitor, he is given the opportunity to include a description of those graphs saved in the Data File which he wants to be used as input data for the particular job he is preparing. The option is also available for the graph being displayed at the DEC-338 to be used as this type of input data. The following subroutine call is used in an interactive ALLA program to read into the ALLA structure one graph from the input data stream:

```
CALL GRAPIN(ent)
```

where ent is a nonsubscripted entity variable. Whereas the GETGRAPH statement may be executed any number of times, this subroutine call may be called only once for each graph in the input data. One extra call is allowed in order for the program to be able to accept an arbitrary number of graphs as input data. The extra call returns with UNDEF as the value of ent. If this ever occurs, no further calls on GRAPIN may be made by the interactive ALLA program.

The graphs which are inputted by either of the above methods are encoded in precisely the same format as a sequence of 12-bit words. The encoding includes all of the information which is available through the status communication cells about each vertex and arc whose internal name is less than 4080 (see Section 4.3.9). In addition each encoded graph includes the following information which is associated with the entire graph:

a) Whether BLINK Mode or DIM Mode prevails

b) If in DIM mode, the dimming intensity level

c) Window Size

d) Overall Picture Intensity Level

e) Window Position

f) Vertex and Arc Created Internal Names

An encoded graph is read into the ALLA structure by the ALLA subroutine GRAPIN. As indicated above, when an interactive ALLA program calls the subroutine, it reads the encoded graph from the input data. The GRAPIN subroutine makes calls on the subroutine GET12 which returns through its one argument a 12-bit word which is next in the sequence of 12-bit words of the encoded graph. The GET12 subroutine is coded in MAP assembly language and normally obtains its input from the input data stream. The GETGRAPH statement is interpreted by another assembly language subroutine which sets a program switch in GET12 and then calls upon GRAPIN to read in the encoded graph. With the alternate setting of the switch, GET12 essentially fetches its input words from the DEC-338. Since the GRAPIN subroutine is such an important part of the Interactive Graph Theory System a listing of it is included in this report in Appendix 5. A functional description of its operation is presented

in the following subsection.

4.7.7.1 Functional Description of GRAPIN

The GRAPIN subroutine, written in interactive ALLA, is used to read into the ALLA structure an encoded graph being used as input in an interactive ALLA program. The subroutine is called with one argument which is an entity variable, whose value becomes the graph being inputted. The graph is defined as a pair with integer properties directly corresponding to status information as returned in IBM 7040 communication cells as follows (see 4.7.4.1 through 4.7.4.4 and 4.7.4.8):

Property Name	Communication Cell
GMISC (bit 24)	BLNDIM (sign bit)
GMISC (bits 25-27)	BLNDIM (bits 33-35) when in DIM mode
GMISC (bits 31-32)	WINSIZ
GMISC (bits 33-35)	INTENS
GYWIND	YWIND
GXWIND	XWIND
CRNAMS (bits 12-23)	ARCCRN
CRNAMS (bits 24-35)	VERCRN

The left element of the pair which is the graph being defined is a set of all vertices of the graph. Each vertex is an atom with integer properties directly corresponding to the status information as returned in IBM 7040 communication cells as follows (see Section 4.7.4.11.1):

Property Name	Communication Cell
GMISC	STAT1
INNAME	STAT2
YCOORD	STAT3
XCOORD	STAT4
GDIM	STAT5
OFFSET (bits 12-23)	YOFF
OFFSET (bit 24-35)	XOFF
LABEL1	LABEL(1)

LABEL2	LABEL(2)
LABEL3	LABEL(3)
LABEL4	LABEL(4)
LABEL5	LABEL(5)

The right element of the pair which is the graph being defined is a set of all arcs of the graph. Each arc is a pair whose left element is the vertex which is the from-vertex of the arc, and whose right element is the vertex which is the to-vertex of the arc. In addition, each arc has integer properties directly corresponding to the status information as returned in IBM 7040 communication cells as follows (see Section 4.7.4.11.2):

Property Name	Communication Cell
GMISC	STAT1
INNAME	STAT2
GDIM	STAT5
OFFSET (bits 12-23)	YOFF
OFFSET (bits 24-35)	XOFF
LABEL1	LABEL(1)
LABEL2	LABEL(2)
LABEL3	LABEL(3)
LABEL4	LABEL(4)
LABEL5	LABEL(5)

The integer properties used in the structure of a graph as described above are not automatically declared within interactive ALLA programs. This implies that the programmer must include INTEGER declarations for each of the graph properties he uses within each subprogram. If a program includes statements to change the value of one of these properties, that property name must also be included in a PROPERTY declaration statement.

4.7.8 Termination

Section 4.7.5 described how the user can stop the interactive execution process by hitting the manual interrupt button. The programmed control of termination is described here. An interactive ALLA statement may be used to terminate interaction in the normal case. A subroutine named ERROR is also included in the Interactive Graph Theory System for the use of the programmer.

4.7.8.1 Normal

The following statement is used in an interactive ALLA program to terminate interactive execution:

```
TERMINATE
```

It causes a special DOGGIE command to be outputted, and this is followed by a flush of the DOGGIE buffer. After execution of the TERMINATE statement, control immediately returns to the next statement. The interactive ALLA program may continue to execute, but no further interactive statements will have any effect. Meanwhile, an indicative termination message has been displayed to the user, and this leads him back to the Graph Monitor.

4.7.8.2 Error

A standardized fatal error subroutine is available to the programmer, particularly for including in algorithms at places where there are what the programmer expects to be "impossible" paths. For example, a THROUGH loop might be used to perform a search, and according to the algorithm a match must be found before the loop is exhausted. For safe programming, the programmer may include a call upon the ERROR subroutine immediately following the range of the loop as a protection against the unforeseen occasion when the search may not yield a match.

The following subroutine call is used in an interactive ALLA program to terminate interactive execution, and display to the user a short error message:

CALL ERROR(text)

where text is a Hollerith constant. The ERROR subroutine is written in interactive ALLA. It begins by outputting four DOGGIE words of 0 in order to terminate a possibly pending DOGGIE command. This is followed by a DOGGIE command to ring the bell on the Teletype twice. Next a message is placed at message line 6 which reads "ERROR IN EXECUTION." The text passed to the ERROR subroutine as an argument is then placed at message line 5. Note that only up to 27 characters may be displayed on one message line. The subroutine ends with the statements TERMINATE in order to cease interaction and STOP in order to cease execution. A listing of the ERROR subroutine is included in Appendix 5.

4.7.9 Helpful Functions

Since both bit handling and code conversion are common operations in interactive ALLA programs, there are four functions available which are part of the Interactive Graph Theory System. Each is an integer function and must be declared in an INTEGER declaration statement in each subprogram where the function is used. Each of these functions is written in MAP assembly language.

4.7.9.1 Bit Handling

Two of the helpful functions are used for handling bits within 36-bit ALLA integer variables. The need for these functions arises in the GRAPIN routine and in places where certain properties of a graph are being used. Each of these functions has three arguments. The first two arguments must be integer expressions whose value is between

0 and 35 in order to signify bit numbers in a 36-bit word. The third argument is an integer expression whose value is being either unpacked or packed. The first argument represents the leftmost bit number, and the second argument represents the rightmost bit number of the sequence of bits of the third argument.

4.7.9.1.1 Unpacking

The following integer function is used for unpacking a sequence of bits of a 36-bit word:

BITS(from,to,word)

where from, to, and word are integer expressions. The value of this function is only those bits specified adjusted to the low-order end of the 36-bit value. For example, if V is an entity variable whose value is a vertex of a graph entered into the ALLA structure by GRAPIN, then the following application of the BITS function yields the y-offset of the label of that vertex:

BITS(12,23,OFFSET(V))

4.7.9.1.2 Packing

The following integer function is used for packing a quantity into a particular sequence of bits of a 36-bit word:

BITSIN(from,to,word)

where from, to, and word are integer expressions. The value of this function is the number of bits specified by the values of from and to taken from the low-order end of the value of word and repositioned at bit positions from through to in the 36-bit value. For example, in the GRAPIN subroutine, the 12-bit quantity of the y-offset is in TEMP, and the x-offset is also a 12-bit quantity in TEMP1. The following statement defines the OFFSET property of an entity:

OFFSET(ENT) = BITSIN(12,23,TEMP) + TEMP1

4.7.9.2 Code Conversion

The other two helpful functions are used to convert a 36-bit word treated as six 6-bit character codes between trimmed ASCII and IBM 9-code. The following integer function converts all characters from ASCII to IBM code:

ATOI(word)

where word is an integer expression. The following integer function converts all characters from IBM to ASCII code:

ITOA(word)

where word is an integer expression. Table 4-5 presents the correspondence between trimmed ASCII and IBM character codes used by these two functions.

4.7.10 Reserved Words

Section 3.18.1 presented a list of words which the programmer of pure ALLA must avoid. With the addition of interactive components to ALLA, the list of reserved words has been extended significantly. Table 4-6 is an alphabetical list of words which the interactive ALLA programmer may not use as his own variable names, function names, or subroutine names. Since DOGGIE words must appear only with DOG and DOGSET statements, they do not impose any restrictions.

4.7.11 Logical-IF Statement

Section 3.18.3 presented a list of statement forms which the programmer of pure ALLA could not use to the right of the logical expression in a Logical-IF statement. With the addition of interactive components to ALLA, the list has been extended to include the following forms:

Table 4-5
Correspondence Between Trimmed ASCII and IBM Character Codes

ASCII (octal)	IBM (octal)	Character	ASCII (octal)	IBM (octal)	Character
00	77	end-of-list	40	60	space
01	21	A	41	52	!
02	22	B	42	17	"
03	23	C	43	75	#
04	24	D	44	53	$
05	25	E	45	37	%
06	26	F	46	32	&
07	27	G	47	14	'
10	30	H	50	74	(
11	31	I	51	34	)
12	41	J	52	54	*
13	42	K	53	20	+
14	43	L	54	73	,
15	44	M	55	40	-
16	45	N	56	33	.
17	46	O	57	61	/
20	47	P	60	0	0
21	50	Q	61	1	1
22	51	R	62	2	2
23	62	S	63	3	3
24	63	T	64	4	4
25	64	U	65	5	5
26	65	V	66	6	6
27	66	W	67	7	7
30	67	X	70	10	8
31	70	Y	71	11	9
32	71	Z	72	15	:
33	35	[	73	56	;
34	76	\	74	36	<
35	55	]	75	13	=
36	57	↑	76	16	>
37	12	←	77	72	?

Table 4-6 Reserved Words in Interactive ALLA Programs

ARCCRN	INTENS	SETVAL
ATOM	LABEL	STAT1
BLNDIM	LELM	STAT2
CHPSEU	LPHIT1	STAT3
CLRPB	LPHIT2	STAT4
CRATOM	LPHIT3	STAT5
CREATE	LPHIT4	STLELM*
CRPAIR	MEMBER	STRELM*
CRSET	MESSAG	TERMIN*
DELETE	NULL	TTYBUF
DOG	PAIR	TTYIN
DOGFLU*	PB	UNDEF
DOGSET	PBS	USEENT
DOGSTR*	POP	USEPR
DOGTEX*	PRBCD	USER
EMPTY	PRENT	USESET
ENTITY	PRNAME	USETYP
ERROR	PRSET	VERCRN
ESCAPE	PRVAL	WAITCH*
FORALL	PUSH	WINSIZ
FORNXT*	RELM	XOFF
FRBLKS	REMOVE	XPSEUD
GETGRA*	REMPRO*	XWIND
GETSTA*	SELECT	YOFF
GRAPIN	SET	YPSEUD
INSERT	SETPB	YWIND

* These words are not explicitly used in interactive ALLA statements, but their use is restricted due to the underlying implementation of the ALLA compiler and execution-time system.

DOG ...

DOGSTRING ...

DOGTEXT ...

DOGSET ...

GETSTATUS

WAITCHANGE

ESCAPE

DOGFLUSH

GETGRAPH ...

TERMINATE

4.7.12 Sample Interactive Program

This section presents a rather small, yet complete example of an interactive ALLA program. It initially places two messages (for the user) at the lower left corner of the screen. As the messages indicate, the program changes the shape of all vertices seen by the light pen to be square (vertex shape 6) until either pushbutton 11 or the manual interrupt button is depressed. Section 4.6.4 presented a sample user program which has the same operational characteristics as this one. A source listing of the sample interactive program follows:

```
      SUBROUTINE SAMPLE
      CALL CLRPB(11)
      DOG START LTPEN WHOLE VERTEX, ALL
      CALL MESSAG(2)
      DOGSTRING 'POINT TO VERTICES TO BE'
      CALL MESSAG(1)
      DOGSTRING 'MADE SQUARE, PB 11 TO STOP'
10    DOG ALLHIT
20    WAITCHANGE
      IF (PB(11)) GOTO 30
      IF (LPHIT1 .EQ. 0) GOTO 20
      DOG START EXIST SHAPE VERTEX 6,(LPHIT2)
      GOTO 10
30    TERMINATE
      STOP
      END
```

CHAPTER 5

OPERATION OF THE TERMINAL

Whenever the Moore School Problem Solving Facility is operating, a user may use the Interactive Graph Theory System as any other user uses the system. Other users may be using Teletype terminals, but the Interactive Graph Theory System must be used from the DEC-338 graphics terminal. Moreover, the minidisk of the DEC-338 must be set up with at least a basic system. Each user of the Interactive Graph Theory System carries a DECtape which includes such a disk image. Less than one minute is needed to load the disk from DECtape, and then the only other consideration is to make sure the DEC-338 Dataphone line is connected to the PDP-8 being used as the intermediary satellite of the Moore School Problem Solving Facility.

5.1 Graph Monitor

At the terminal, the user initializes by typing a very brief loading command to the PDP-8 Disk Monitor (five typed characters), and DOGGIE is loaded and started. The Graph Monitor is automatically started, and the display screen appears as in Figure 5-1. The Graph Monitor is the user program segment which allows the user to select one of the nine alternatives displayed on the screen. The Graph Monitor is the basic starting point for all operations at the terminal using the Interactive Graph Theory System. The user may return back to the Graph Monitor at any time by hitting the large button on the pushbutton box labeled "INTERRUPT".

When the Graph Monitor operates, the user is expected to select one of the nine functions displayed. To the left of each function is displayed a small triangular light button and a digit. To make a

SELECT ONE OF THE FOLLOWING

▲1 CREATE	▲4 SAVE	▲7 CHANGE WINDOW
▲2 ALTER	▲5 RESTORE	▲8 TEXT CONSOLE
▲3 REMOVE	▲6 EXECUTE	▲9 MISC FUNCTIONS

Figure 5-1 Graph Monitor

selection, the user may either use the light pen to point at a triangle or he may type the associated digit at the Teletype. The effect of either selection method is the same, which is the presentation of informative display messages, which in general give further choices for the user to indicate the type of action he is seeking. Each of the nine possible message displays after the Graph Monitor are unique and indicate which choice was made in case the light pen pointing happened to be sloppy. For this same reason, no irrevocable operation is performed as a result of a first choice. If the user decides that the wrong choice had been made, he may simply push "INTERRUPT" to get back the Graph Monitor.

The following sections describe the various operations which can be done at the DEC-338. Most of these constitute a general graph drawing facility which is entirely local to the terminal. This includes management of the window on the paper, and a complete facility for alteration of the graph including arbitrary repositioning of vertices. The following section gives more detail about the actual interactions which occur at the terminal than the succeeding sections. This detail is included for the reader who does not have the opportunity to either use the system or watch it being used. Even so, there is much about the way the terminal "feels" to a user which is important and yet cannot be easily described in writing.

5.2 Create

When the user selects the first alternative available under the Graph Monitor, the display screen appears as in Figure 5-2, thus requiring specification of the particular type of creation to be done. One consideration which arises as these choices are displayed is that the light buttons of the new message should be sufficiently far from

SELECT WHAT TO CREATE
▲ VERTEX
▲ ARC LABEL
▲ ARC
▲ VERTEX LABEL
▲ ARROW

Figure 5-2 Create Options

the location on the screen where the light pen may remain after pointing at the previously displayed message.

Initially, when there are no existing graph parts, the only reasonable choice is the creation of a vertex. After the user points to the corresponding light button, the screen appears as in Figure 5-3. Note that again, he may retract his choice one level by depressing pushbutton 5. This possibility is available after each of the five choices of the creation operation. Of course, the user may also revert back to the Graph Monitor by depressing "INTERRUPT". A graphical terminal provides an effective medium for correcting errors in operation. This system takes advantage of that feature, thus being rather forgiving.

From the state of the screen as in Figure 5-3, the user may use the light pen to position the cursor and pseudo-pen-point shown in the center of the screen. After positioning, he depresses pushbutton 11 to create a vertex and the screen then appears as shown in Figure 5-4. He now has the choice of altering the shape of the freshly created vertex or labeling it. In order for the user to be sure which vertex he just created it is blinking at this time. A light button is used for the former and the latter option is accomplished by typing on the Teletype keyboard. Figure 5-5 shows the display screen after the vertex has been labeled. The user may then depress pushbutton 11 again to be given the facility to create another vertex. As this is done blinking finally stops on the previously created vertex. This process continues until more labeled vertices have been created as shown in Figure 5-6.

PB 11 CREATES VERTEX AT PEN

OR PB 5 TO CREATE MORE

Figure 5-3 Create a Vertex

YOU MAY TYPE A LABEL
▲POINT HERE TO CHANGE SHAPE
PB 11 TO CREATE MORE

Figure 5-4 First Vertex Created

● VERTEX 1

YOU MAY TYPE A LABEL
△ POINT HERE TO CHANGE SHAPE
PB 11 TO CREATE MORE

Figure 5-5 First Vertex Labeled

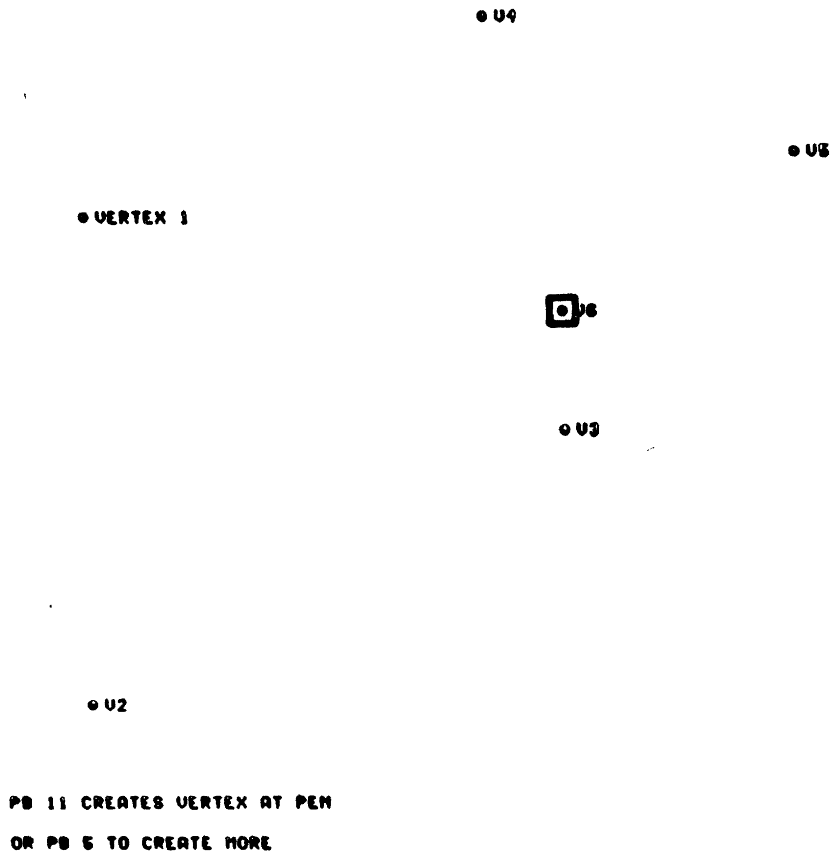


Figure 5-6 More Vertices

Next, when the user depresses pushbutton 5, he is again given the choice of what to create as shown in Figure 5-7. Now if he selects the light button labelled "ARC", the display screen indicates the first step in the creation of an arc. Figure 5-8 shows the instructions to the user to point to a from-vertex. The interaction which follows is typical of many of the graph manipulation sequences which are available under the Graph Monitor. It is also used in the SELECT interactive user program as described in Section 4.7.6.2.2.1. When the user points to a vertex which is to be the from-vertex of the arc, the vertex blinks and the instructions to the user indicate that the blinking vertex will be accepted only when pushbutton 11 is depressed. Until then, the user may point to another vertex as a tentative candidate. This feedback is particularly helpful when the user is selecting an entity which is rather close to another selectable one. Since a displayed graph may be located anywhere on the screen, a pushbutton rather than a light button must be used to indicate the final choice.

After a from-vertex has been selected, the user may either select a to-vertex by pointing or use a pushbutton to indicate he wants to create a loop. In either case the same selection procedure is used. Figure 5-9 shows the display screen after the user has created an arc. He is given the options of making the arc a directed one by displaying an arrowhead and he may also label it. Figure 5-10 shows the screen after more labelled directed arcs have been created, and control has returned to the Graph Monitor.

The remaining options of the creation function permit the user to display an arrow on an arc and possibly reverse its direction (if it is not a loop), and to create a label for a vertex or arc.

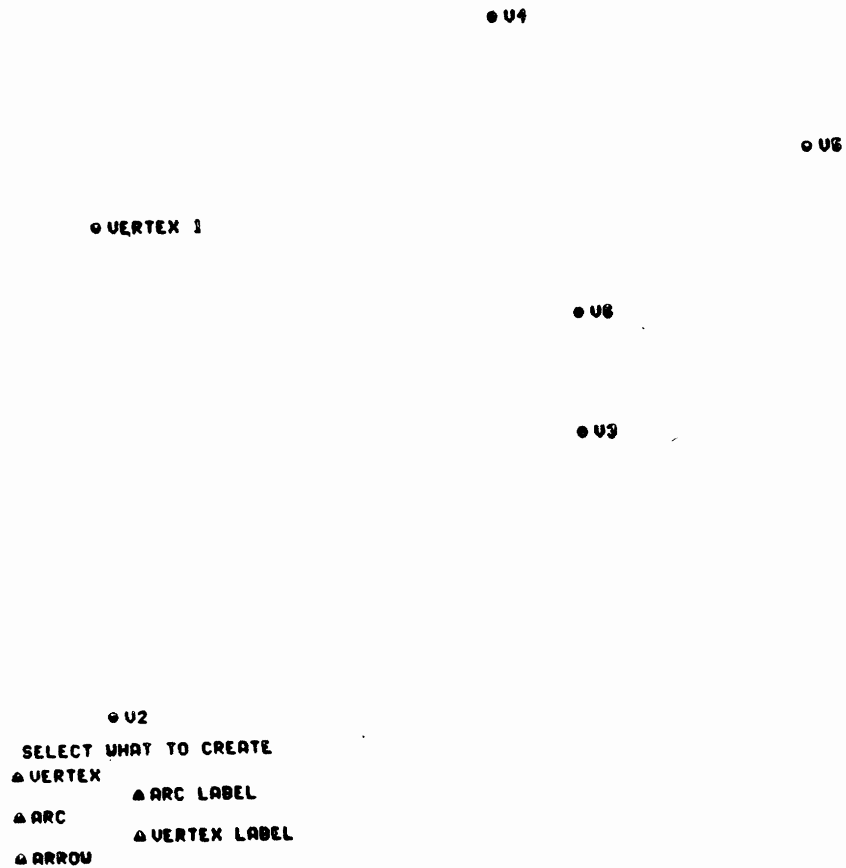


Figure 5-7 Ready to Create Arcs

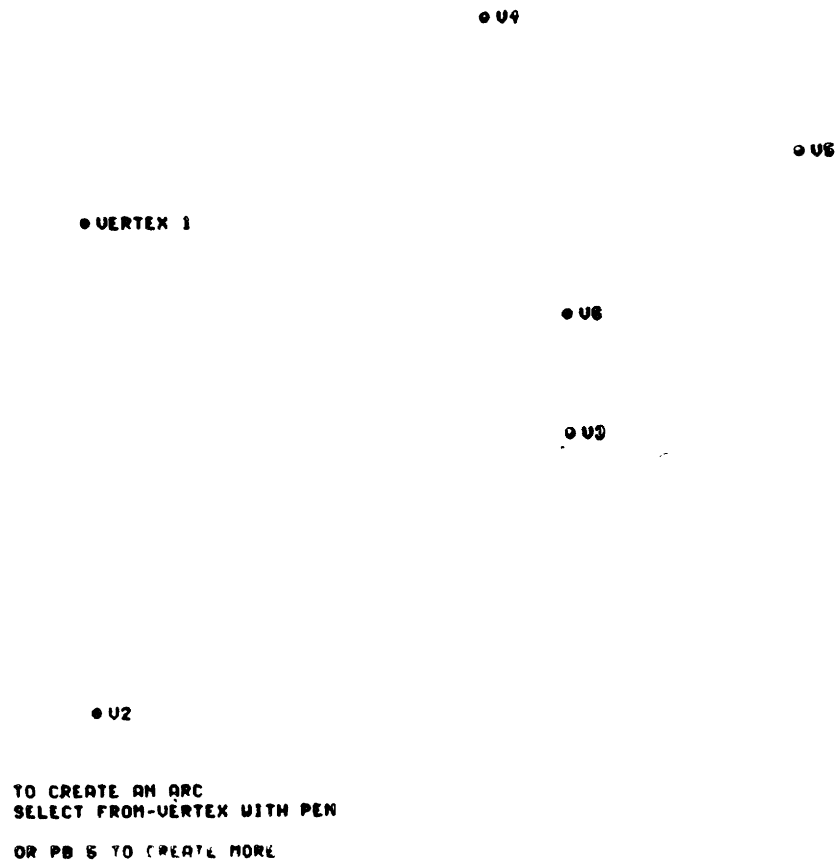
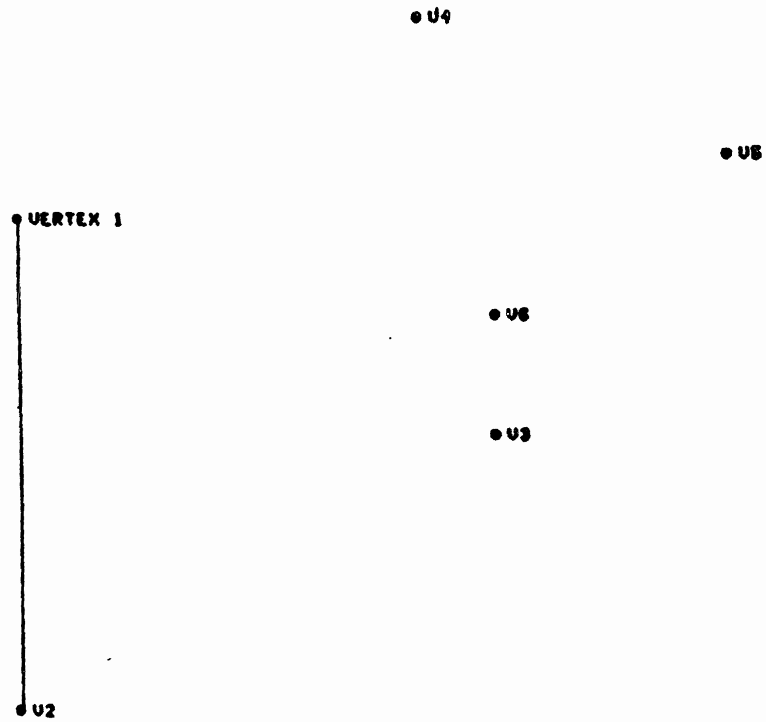
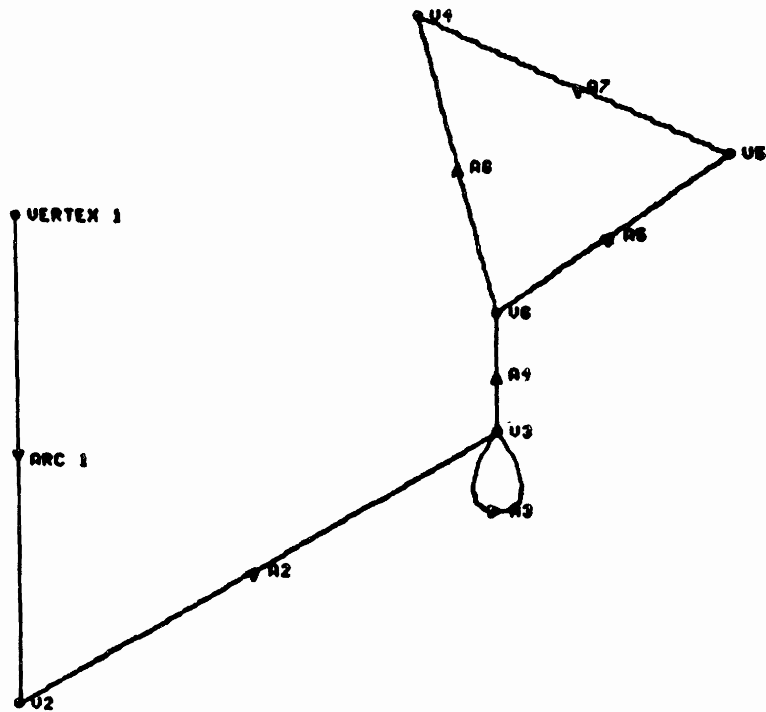


Figure 5-8 Creating First Arc



YOU MAY TYPE A LABEL
PB 10 ON FOR ARROW
PB 11 TO CREATE MORE

Figure 5-9 First Arc Created



SELECT ONE OF THE FOLLOWING

▲1 CREATE	▲4 SAVE	▲7 CHANGE WINDOW
▲2 ALTER	▲5 RESTORE	▲8 TEXT CONSOLE
▲3 REMOVE	▲6 EXECUTE	▲9 MISC FUNCTIONS

Figure 5-10 More Arcs

5.3 Alter

The second alternative available under the Graph Monitor permits the created graph to be altered. When the user selects this alternative the screen appears as in Figure 5-11. Under vertex alteration, either a vertex shape may be altered or a vertex may be moved. The latter option is most impressive to perform since as the user drags a chosen vertex around on the screen all connected arcs are continuously adjusted to remain attached. This effect is usually described as "rubber-banding". As the arcs change orientation their associated arrows are also kept pointing along the proper direction. Figure 5-12 shows the display screen after two of the vertices have been moved.

There are two ways in which arcs may be altered. First, an arc may be moved, which means that it may be redefined to have any other from-vertex or to-vertex. Figure 5-13 shows the screen after the arcs labelled "A2" and "A3" have been moved. The second type of arc alteration is the feature of bending an arc. This operation introduces an extra joint in an already existing arc by actually creating a vertex of shape number 1 in the middle of the arc. The user may move this bend anywhere on the screen by the light pen. Figure 5-14 shows the display screen after a bend has been introduced into the arc labelled "A2".

Alteration of an arrow amounts to allowing the user to reverse its direction.

Vertex and arc labels may be altered in two ways: an entity may be given a new text label, or the label may be moved to some other position relative to the entity. Figure 5-15 shows the first vertex and arc relabelled with "V1" and "A1"; the labels "V6" and "A7" have been moved, and the Graph Monitor has been restarted.

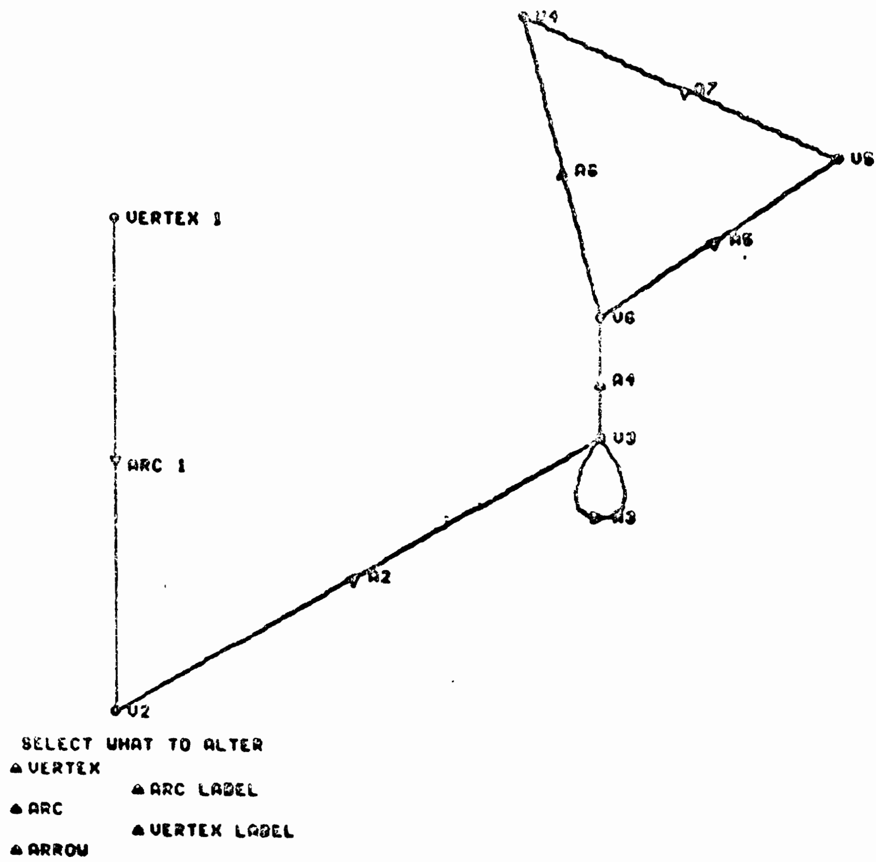
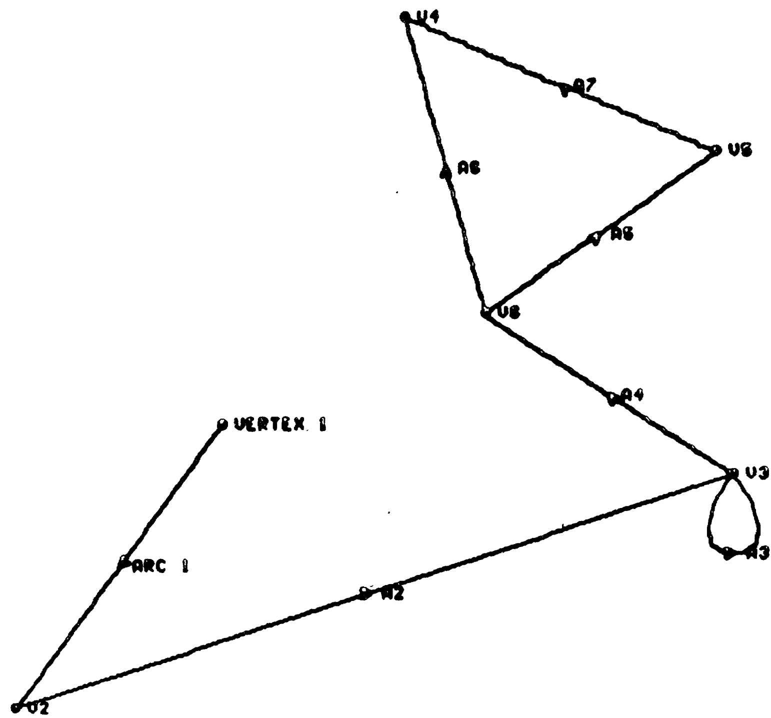
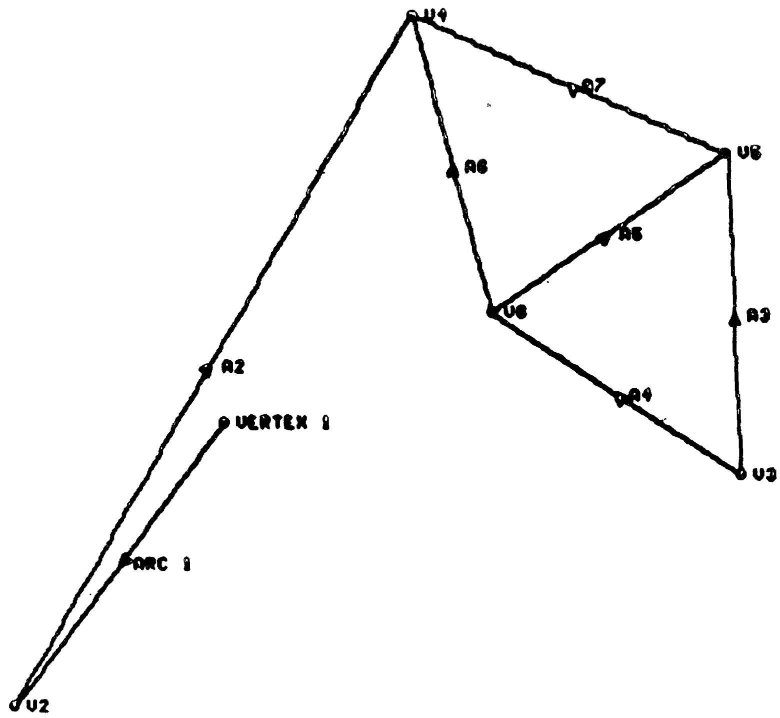


Figure 5-11 Alter Options



POINT TO VERTEX TO BE MOVED
OR PB 5 TO ALTER MORE

Figure 5-12 Two Vertices Moved



POINT TO AN ARC TO BE MOVED

OR PB 5 TO ALTER MORE

Figure 5-13 Two Moved Arcs

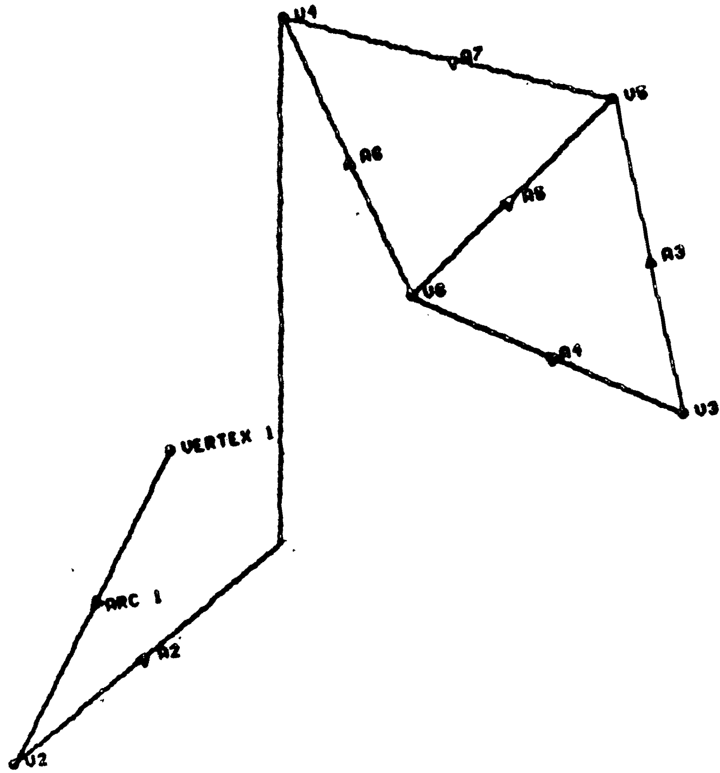
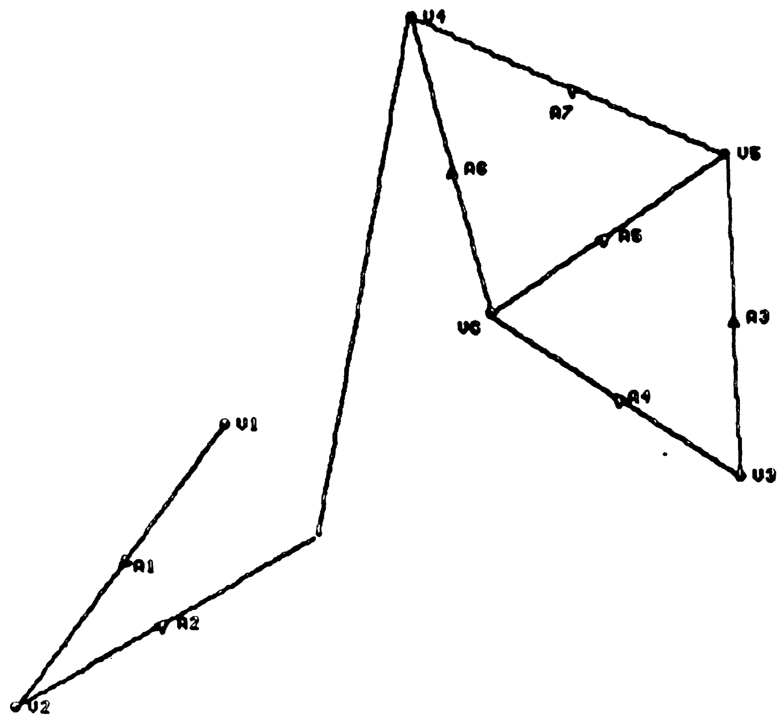


Figure 5-14 A Bent Arc



SELECT ONE OF THE FOLLOWING

1 CREATE	4 SAVE	7 CHANGE WINDOW
2 ALTER	5 RESTORE	8 TEXT CONSOLE
3 REMOVE	6 EXECUTE	9 MISC FUNCTIONS

Figurè 5-15 Label Alteration

5.4 Remove

The third alternative available under the Graph Monitor permits the removal of any part of the displayed graph: vertices, arcs, arrows, and labels. When a vertex or arc is removed, its associated parts are also removed. When a vertex is removed all arcs which are connected to that vertex disappear. (They are still defined in the structure, but cannot be displayed.) Figure 5-16 shows the same graph with the following parts removed: vertex "V1", and therefore arc "A1", arc "A5", the arrow of arc "A4", and the labels of vertex "V4" and arc "A3". Also shown in this figure are instructions to the user for the removal of an arc label.

5.5 Save

This alternative is the fourth one available under the Graph Monitor. It permits a user to take a snapshot of the graph defined as being displayed on the paper. The graph is encoded into a compact sequence of 12-bit words and saved in the MULTILIST data file on the disk of the IBM 7040. It is saved with from one to six alphanumeric keys (descriptors) which the user must type on the Teletype keyboard. Before the graph is saved, any graph already in the file is first deleted if it is described (at least) by all keys being used to describe the new graph. Therefore each saved graph must have a unique description. In order for the graph to be saved, a MULTILANG job must be executed on the IBM 7040. The DEC-338 automatically sets up this job in proper format, and it must wait in the job queue along with other console jobs. The terminal is temporarily unusable for any other operation until the execution of the job is complete; when this

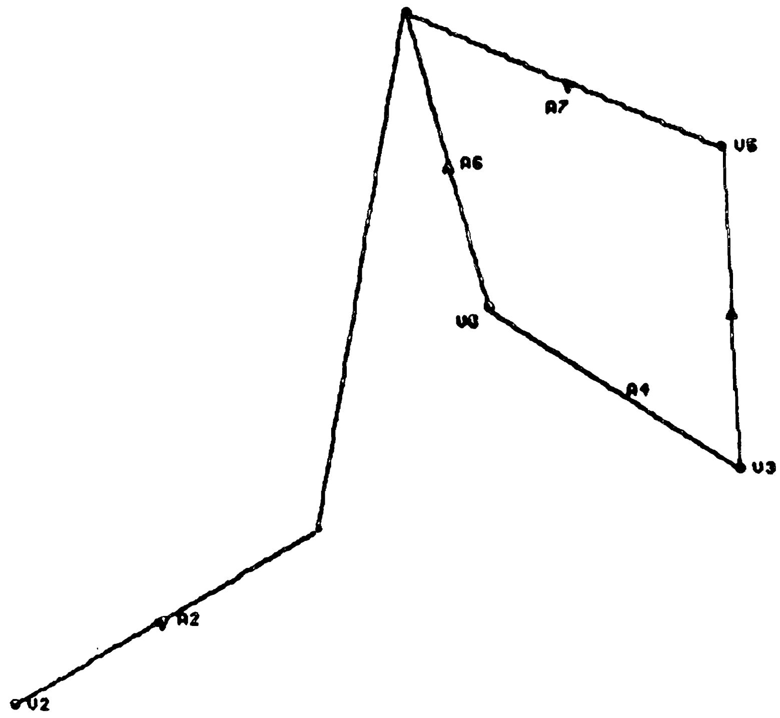


Figure 5-16 Parts Removed

happens the Teletype bell rings. Saving a graph is not a destructive process. Saved graphs may later be restored by description using the operation described in Section 5.6, and they may be included in the input stream of an interactive execution as described in Section 5.7.

5.6 Restore

The fifth alternative available under the Graph Monitor gives the user the facility to restore to the DEC-338 terminal any graph saved away in the MULTILIST data file. Although only one graph may be restored to the paper at any one time, any number of graphs may be retrieved from the MULTILIST data file into an intermediate "Restore File." Then the user may sequence through the list by pushbutton control. The Restore File is preserved as long as no further SAVE, RESTORE, or EXECUTE functions are performed. Therefore, when the user selects the RESTORE option, he is given the choice of freshly starting by restoring some graphs to the Restore File and seeing the first one, or he may request the first graph of the Restore File. In the first case, he is requested to type a description of those graphs to be restored. This may be any logical combination of keys including the operators 'AND', 'OR', and 'NOT'. In order for a retrieval to be performed, a MULTILANG job must be executed on the IBM 7040. The DEC-338 automatically sets up this job in proper format, and it must wait in the job queue along with other console jobs. The terminal is temporarily unusable for any other operation until the execution of the job produces either the first graph or no graphs; the Teletype bell rings when the terminal is available.

Once the Restore File has been filled as a result of a retrieval, graphs may be obtained essentially instantaneously; it is not necessary to execute another job. When any graph restoration is requested there are only two possible responses. If there is a graph to be restored, it is, and then the user is given the choice of returning to the Graph Monitor, restoring the next graph of the Restore File, or restoring the first graph of the Restore File. On the other hand, if there is no graph to be restored an indicative message is displayed and the user may request the first graph of the Restore File or he may return to the Graph Monitor.

5.7 Execute

The sixth alternative available under the Graph Monitor is used to initiate or resume the execution of interactive ALLA programs at the IBM 7040. When the user selects the EXECUTE option, he is then requested to point to a light button to resume execution or type a 'RETURN' on the Teletype keyboard to begin. Resuming execution may be done only after interaction had already been in progress and the interactive ALLA program executed the ESCAPE statement (see Section 4.7.6.1). This causes the direct control of the DEC-338 by the IBM 7040 to be temporarily suspended. The Graph Monitor is placed in control in the DEC-338. The user may take advantage of the local features of manipulation, but he must not attempt to use the SAVE or RESTORE options. Meanwhile the IBM 7040 hangs waiting for the DEC-338 to send a status message; this happens when the user "resumes execution."

When the user chooses to begin execution, he must type a description of those program decks (subroutines and functions) which are to be used for the particular interactive ALLA job. He need not specify

the standard system programs (e.g. GRAPIN) since those are always automatically loaded. The details of the loading are covered in Appendix 5. The description of program decks may be any logical combination of keys including the operators 'AND', 'OR', and 'NOT'. All binary decks in the MULTILIST data file whose description corresponds to the request will be retrieved for loading by the IBSYS loader. If no decks meet the description, the user will be informed of the error condition. When more than one deck with the same name is in the MULTILIST file, the most recently stored version will be used; this is not considered to be an error.

Next, the user is asked to type an entry point name. This must be a deck name or subroutine name included in the program decks requested; it may be from one to six alphanumeric characters.

The third request to the user is for a description of any graphs he wants to be used in the data input stream for the interactive ALLA program. He may type a 'RETURN' to specify none, or he may type a description as any logical combination of keys including the operators 'AND', 'OR', and 'NOT'. All saved graphs in the MULTILIST data file whose description corresponds to the request will be retrieved for the data input stream. If no graphs meet the description, the user will be informed of the error condition. In addition, the user may include the graph currently displayed on the paper to be included at the end of the data input stream. This is specified by typing an UP-ARROW anywhere before the terminating 'RETURN'. Graphs placed on the data input stream may be read into the ALLA structure within an interactive ALLA program by a call upon the GRAPIN subroutine described in Section 4.7.7.

After the user types the 'RETURN' terminating the description (if any) of graph data, the DEC-338 automatically sets up a MULTILANG job in proper format, and it must wait in the job queue along with other console jobs. The terminal is temporarily unusable for any other operation until the execution of the job produces either an error condition or properly begins. The user is kept informed of the progress of his job as it begins loading and as it begins execution. He need not stare at the display screen, however, since the Teletype issues appropriate characters as displayed messages change. If all goes well, the final message displayed is 'EXECUTION BEGUN'. Otherwise, an error due to any one of many conditions causes an indicative message to be displayed. The user is then given the option of immediately using the TEXT CONSOLE to determine the source of the error. This is further discussed in Section 5.9.

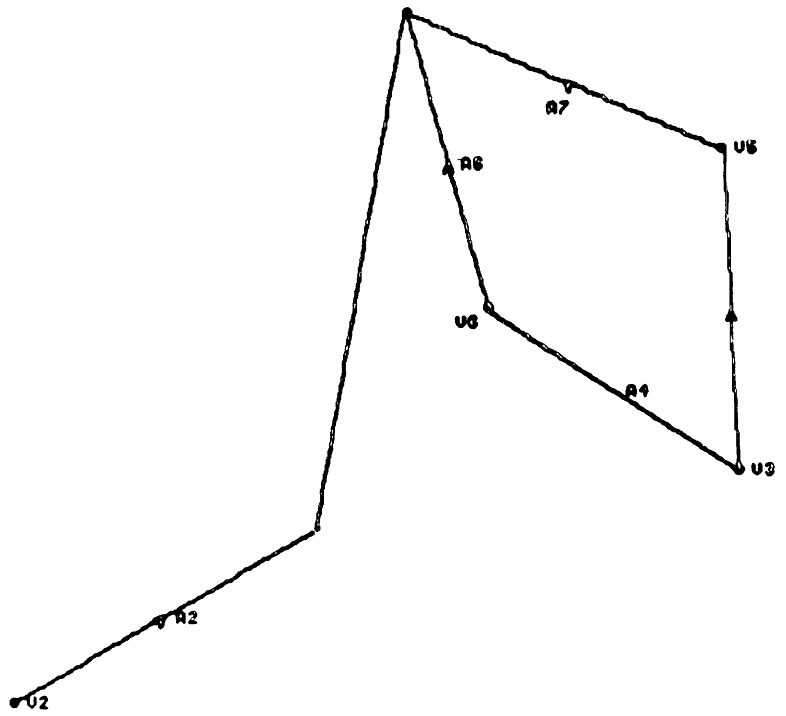
During interactive execution the nature of the operation of the terminal is dependent on the particular interactive ALLA program in control. If the program terminates by itself, the user is appropriately informed. If the user wishes to manually stop interactive execution, he may at any time depress the 'INTERRUPT' button. If he decides to terminate execution at a time when he is using the Graph Monitor as a result of an ESCAPE, he must first resume execution and then use the 'INTERRUPT' button.

5.8 Change Window

With the seventh alternative available under the Graph Monitor the user is given local control of the window being used to view the paper. Light buttons are available which allow the user to select the next smaller or next larger window whenever appropriate. Impossible

choices are not presented; for example, there is no light button displayed which allows for a smaller window when the window size is already the smallest. When the window is not FULL, the user is given the choice of moving the paper under the window frame by light pen control. In any window size, the user is given the option of selecting a window frame size on the paper and positioning any non-FULL window frame anywhere on the paper by light pen control.

Figure 5-17 shows the display screen after the user has selected the CHANGE WINDOW option under the Graph Monitor. Since the window size is HALF, all four options are available to the user. Figure 5-18 shows the effect of choosing the next smaller window. Next, Figure 5-19 shows the display screen after the user chose the option to position the window on the paper. The graph is displayed with FULL window and three possible smaller frame sizes are available. A frame is selected by pointing with the usual method of blinking followed by a confirming push of pushbutton 11. Figure 5-20 shows the screen after the smallest window frame has been selected. Next, the user may drag the displayed window frame around on the screen until he has positioned it where he wants it (see Figure 5-21). Then, when pushbutton 11 is depressed, the screen becomes the chosen window, as shown in Figure 5-22. The option of moving the window is then selected as shown in Figure 5-23. The user may use the light pen to pull the entire paper under the window, as Figure 5-24 indicates after the light pen has been used to pull the cursor (and therefore the paper) upward and to the left.



SELECT ONE OF THE FOLLOWING

△ SMALLER WINDOW △ POSITION WINDOW ON PAPER

△ LARGER WINDOW △ MOVE PAPER UNDER WINDOW

Figure 5-17 Change Window Options

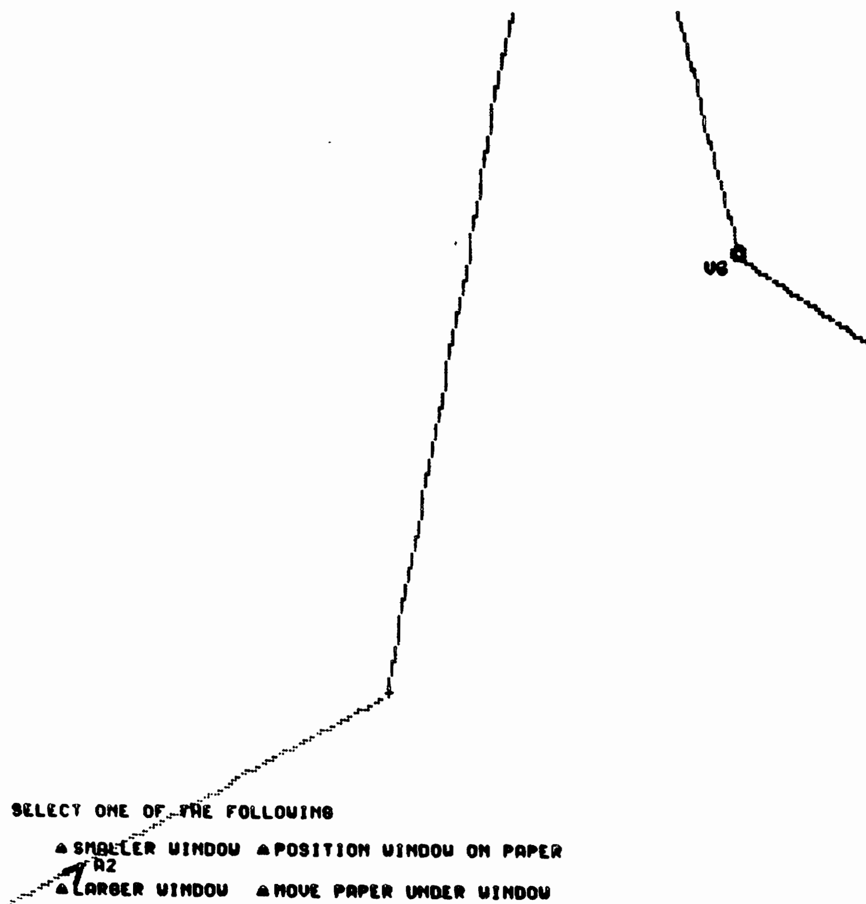
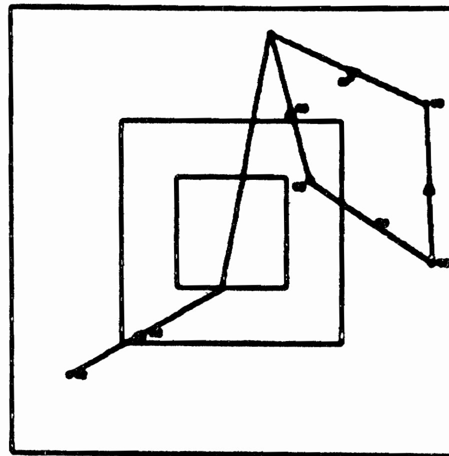
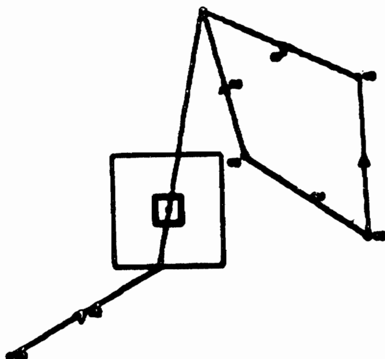


Figure 5-18 FOURTH Window



SELECT FRAME WITH THE PEN
OR PB 11 FOR FULL WINDOW

Figure 5-19 Position the Window



USE THE PEN TO MOVE FRAME
PB 11 FOR FRAME AS WINDOW

Figure 5-20 Smallest Window Frame Selected

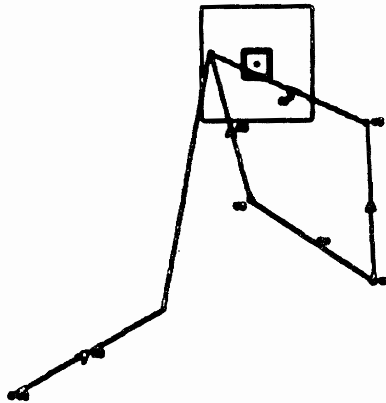


Figure 5-21 Window Frame Positioned

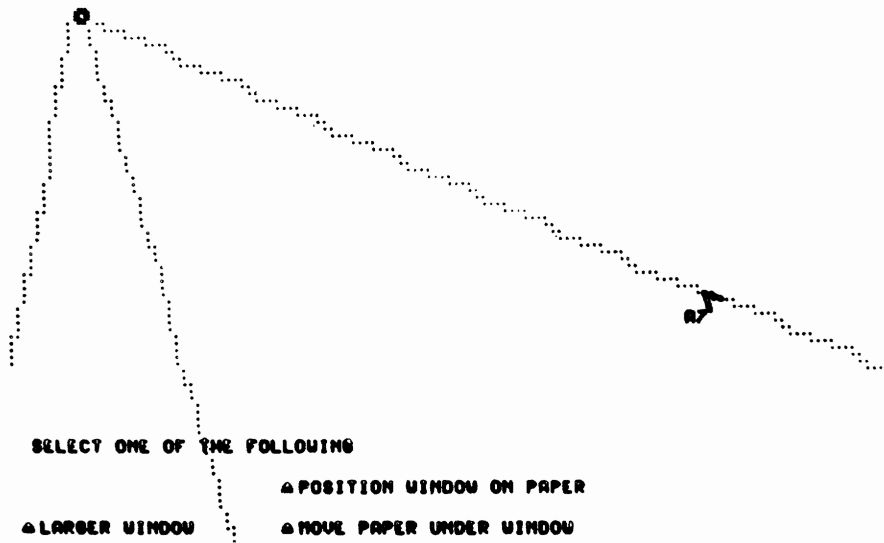


Figure 5-22 EIGHTH Window as Chosen

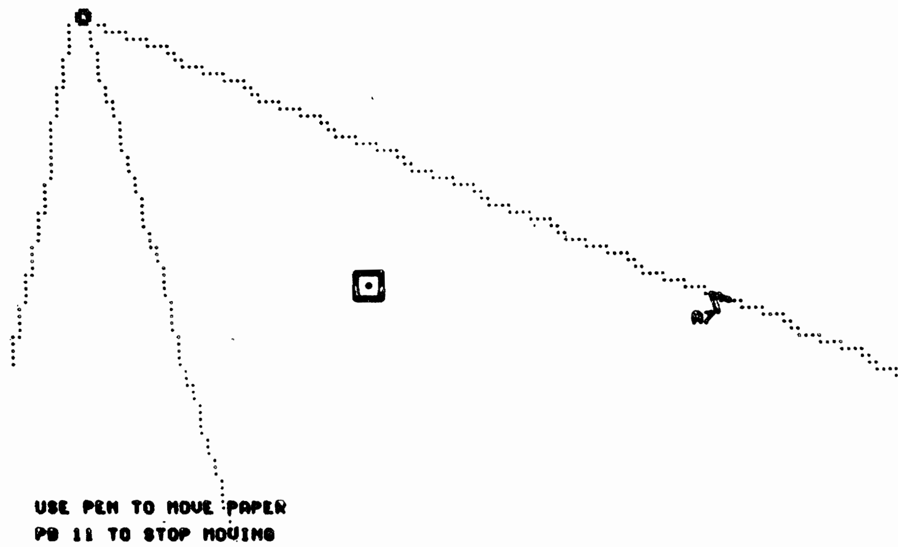
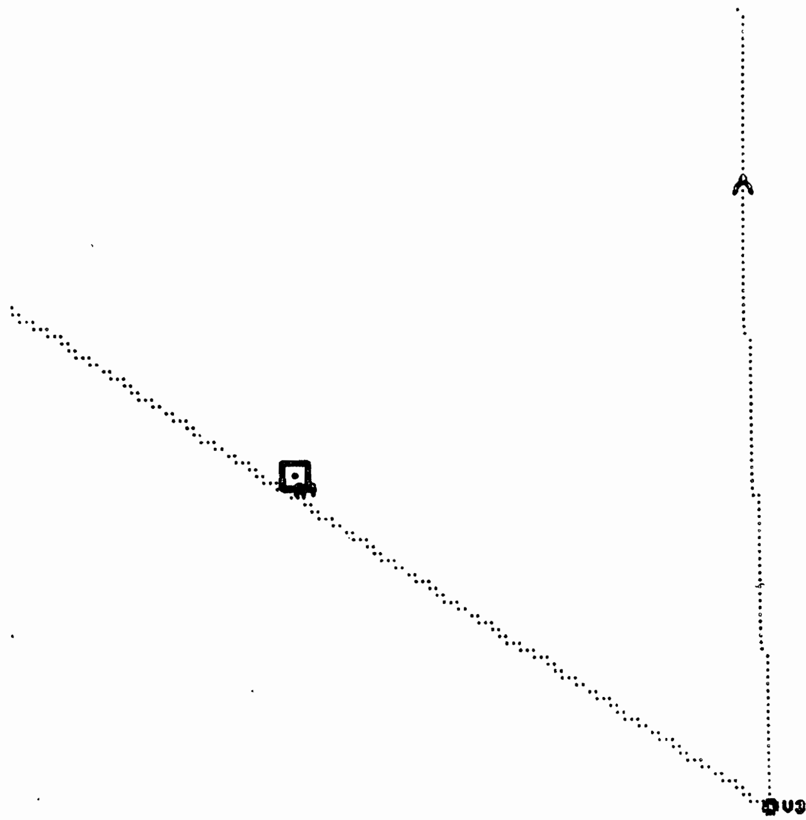


Figure 5-23 Paper to be Moved



USE PEN TO MOVE PAPER
PD 11 TO STOP MOVING

Figure 5-24 Paper Moved

5.9 Text Console

The eighth alternative under the Graph Monitor provides the user with the facility for interaction with IBM 7040 as an alphameric display console. When the user selects this option, there is one more step he must perform as a protective interlock. The user may either depress pushbutton 11 or type 'RETURN' on the Teletype keyboard in order to initiate the text console option. When this happens, any graph existing within the DEC-338 is lost. Later, when the user is through using the text console he may type CTRL SHIFT M which causes the terminal to return to the Graph Monitor in an initialized state with no graph existing.

When the user initiates the text console it is directed toward console number 1. He should use this console for running MULTILANG programs, ALLA compilations, PDPMAP assemblies, saving programs on the disk of the DEC-338, and any general editing or data manipulation operations as are necessary. In the event the text console is being used to discover errors in execution (as mentioned in Section 5.7), the user must direct attention toward console number 2, which is the console where interactive execution is performed.

Use of the text console, with the various console commands is not explained in this report since the details of its operation are independent of the Interactive Graph Theory System. The user is referred to other memoranda and reports which discuss the use of the text console and the way in which it fits into the framework of the Moore School Problem Solving Facility.

5.10 Miscellaneous Functions

The ninth alternative under the Graph Monitor gives the user an extendable menu of choices according to what user programs are on the DEC-338 disk. When the user selects this alternative, display of the currently defined graph is temporarily stopped, and a menu of program segment names is presented. These names are at most four characters in length since they are being taken from the directory of the DEC-338 Disk System. As soon as a user adds a new user program segment to the disk, it will appear as a choice under the MISC FUNCTIONS alternative.

The selection method for operation of a particular program segment is a departure from the standard method of blinking the choice. Instead, as the user points to a light button next to a name, that name brightens to indicate it has been selected. The user may point to another which causes that name only to be brightened. When the user has made a choice, depressing pushbutton 11 causes the selected program segment to be loaded and started. Display of the current graph is also resumed. Fugure 5-25 shows this function with four program segments which the user may call. Note that each name has an associated light button plus one alphabetic character to the left. An alternate method of selection is available for the user: instead of pointing and pushing a button, the user may indicate his choice by typing the corresponding single letter on the Teletype keyboard.

The functions which may be selected by this option are local user programs which are not considered fundamental enough to be included as standard constituents of the system. Many useful local functions have been implemented, however, and some of these are described in the following subsections. As new local functions are implemented, it is

MISCELANEOUS

USER FUNCTIONS

A● DOGI

B● INAM

C● FSAI

D● XCBL

SELECT A PROGRAM

PB11 TO BEGIN EXECUTION

Figure 5-25 Miscellaneous Functions

a trivial matter for any user to add a new program segment to his copy of the DEC-338 disk. The program PIP (Peripheral Interchange Program) may be used for this purpose or to delete unwanted program segments from the disk.

5.10.1 DOGGIE Interpreter (DOGI)

The user program DOGI was originally written to aid in debugging DOGGIE, but it has since been proven useful as a demonstration and learning aid so it remains available. The program enables a user to compose DOGGIE commands on the Teletype in the syntax used to describe the language in Section 4.4. The symbol table of the program includes all DOGGIE words as given in Table 4-2. As the user types a command, each word is checked; if the program does not recognize an input word, it ignores it and responds with "?". Unacceptable characters are ignored and cause the Teletype bell to ring.

A user may compose a sequence of commands by using "," as a separator between two successive commands. Interpretation does not occur until a "RETURN" is typed. The buffer used to hold the 12-bit words being composed is 100 words long. In order to prepare commands which exceed the width of the Teletype carriage, the user may type "VERTICAL TAB" (ctrl K) to indicate continuation. This causes a "-" to be typed followed by an indented new line. Since "-" is not a legitimate input character, the typed copy is not ambiguous.

Each expression typed, delineated by ",", may consist of any number of terms. Each term may be either a DOGGIE word or an octal number. Only the last four digits of an octal number are taken to be the value of such a term. The user may cause the program to ignore the command sequence he is typing by hitting either "ALT MODE" or

"RUB OUT" at any time. The program responds by typing "#" followed by a new line for a fresh start.

Since a "RETURN" cannot be used as one of the characters of a label, the program will automatically type a closing "at-sign" and proceed to interpret when the user types a "RETURN" in the midst of a character string.

The "INTERRUPT" button may always be used to release control and restart the Graph Monitor. In addition, the user may type "ctrl E" in order to end or exit.

5.10.2 Display Internal Names (INAM)

When using DOGI (see Section 5.10.1), it may be helpful to know the internal names of certain vertices or arcs being displayed. The INAM user program is used to label any one vertex or arc or all vertices or all arcs with a 4-digit octal number corresponding to the internal name of the entity. This is accomplished through user selection by light button and pushbutton control.

The "INTERRUPT" button is used to cease the labeling process and restart the Graph Monitor.

5.10.3 Paper Tape Storage of Graphs (XCGL)

Many research projects at the University of Pennsylvania use the IBM 7040 for certain systems which are incompatible with the Moore School Problem Solving Facility. Therefore, the multi-console operating system which is needed for the major computations of the Interactive Graph Theory System is not always available. This restricts any work with graphs to remain local at the DEC-338. In this mode all of the graph drawing and manipulation facilities are available. This use of the local system motivated the writing of a program which allows

the saving of a graph on paper tape. The saved information directly corresponds to the encoding used by the SAVE function described in Section 5.5. Since paper tape has only eight information channels, a compressed binary format is used where three lines of tape correspond to two 12-bit information words.

A corresponding loader program has also been written which reads a paper tape punched in the compressed binary format for graphs and re-creates the saved graph.

The punch program and loader program have been tied together with a small monitor, named XCGL (for Extended Compressed Graphical Loader), which gives the user light button and pushbutton control over loading and punching graphs. The "INTERRUPT" button may be used to restart the Graph Monitor.

5.10.4 Finite State Acceptor Interpreter (FSAI)

Although the IBM 7040 is supposed to be the computer used for graph-theoretic computations, the FSAI user program has been written to demonstrate the power of the small machine. The program interprets a state graph prepared in a prescribed format and carries out the operations of a finite state acceptor using the Mealy model. Input symbols are taken from the Teletype keyboard or reader, and symbols (or even strings) are outputted at the Teletype printer (and punch if desired).

The FSAI program interprets a graph which is of any connectivity, but has a unique starting state indicated by a special shape - the largest vertex shape (number 7). All arcs are assumed to be directed independently of whether their arrows are displayed. Each arc represents a possible transition from one state (vertex) to another.

A label may be of the form:

$$X_1, X_2, \ldots, X_n \qquad \text{or}$$

$$X_1, X_2, \ldots, X_n / Y_1 Y_2 \ldots Y_m$$

where each X is either a single alphameric character or the special terms "LET" or "NUM". "LET" is a shorthand for any alphabetic character, and "NUM" represents any digit. The Y's are any characters.

The program begins by blinking the starting state. When a character is typed (or read) each arc directed out of the current state is checked. Each X or each arc label is checked until a match occurs. In this case a transition occurs to the new state to which the corresponding arc points. If the label included some Y's, they are typed out. Blinking of the previous state stops, and the new state blinks to indicate the current state. If there is no match on any labeled arc, then if there is an unlabeled arc leaving the current state, it is taken; otherwise, the state remains unchanged as if there had been no input. The process continues as each input character is given.

The "INTERRUPT" button is used to stop the process and restart the Graph Monitor.

CHAPTER 6

APPLICATIONS

6.1 Introduction

This chapter presents examples of the application of the Interactive Graph Theory System to three different types of problems in order to demonstrate various properties of the system. Each example includes program listings, pictures of the display screen, and explanatory text. The first example finds the shortest path between any pair of vertices in a weighted directed graph. It demonstrates the use of data associated with arcs and a method for indicating particular subparts of a graph. The second example demonstrates more interactive methods where graph connectivity and vertex positioning are used by the algorithm in order to produce an aesthetic presentation of a tree. The third example is the solution of a purely graph-theoretic problem which produces an arbitrary number of graphs as a solution. All three examples serve as further demonstrations of the use of the interactive ALLA language.

6.2 Shortest Path

Given an arbitrary directed graph the problem is to compute a shortest path from a selected starting vertex to a selected ending vertex. A non-negative integer weight or cost is associated with each arc of the graph. The cost of a particular path in a graph is the sum of the costs associated with the arcs comprising the path. The term "shortest path" is used to indicate a minimal cost route. This problem has a wide variety of applications particularly in networks of communica-

tions or transportation systems.

A solution of the problem has been programmed for the Interactive Graph Theory System with interactive user selection of the starting vertex and ending vertex. If there is no possible path, the program will so indicate; otherwise, a shortest path is displayed as a set of bright arcs against a background of the dimmed graph. The user is then given a choice of either choosing another pair of vertices or arbitrarily altering the graph before another path is requested. Alteration would typically consist of assigning new weights or even changing the structure of the graph.

One entity function named SHPTHW (for shortest path, weighted) computes the path given a graph with each arc having an integer property WEIGHT, and given the starting vertex and ending vertex. The value of the function is a set of arcs which comprise a shortest path. If no path is possible, the set is empty. A source listing of the SHPTHW function follows.

```
         ENTITY FUNCTION SHPTHW(A,B,G)
         ENTITY A,B,G,DSET,TDSET,V,OAV,RVOAV,AA
         LOGICAL SWITCH
         ENTITY INARC,OUTARC
         INTEGER WEIGHT,DIST
         PROPERTY DIST
         THROUGH 10 FORALL V IN LELM(G)
   10    DIST(V) = 10000000
         DIST(A) = 0
         INSERT A INTO CRSET(DSET)
   20    TDSET = DSET
         CREATE SET DSET
         THROUGH 40 FORALL V IN TDSET
         THROUGH 30 FORALL OAV IN OUTARC(V)
         RVOAV = RELM(OAV)
         IF (DIST(RVOAV) .LE. (DIST(V) + WEIGHT(OAV))) GOTO 30
         DIST(RVOAV) = DIST(V) + WEIGHT(OAV)
         INSERT RVOAV INTO DSET
   30    CONTINUE
   40    CONTINUE
         DELETE TDSET
         IF (.NOT.EMPTY(DSET)) GOTO 20
         DELETE DSET
         CREATE SET SHPTHW
         IF (DIST(B) .EQ. 10000000) GOTO 300
         V = B
  110    IF (V .EQ. A) GOTO 300
         THROUGH 120 FORALL AA IN INARC(V)
  120    IF ((DIST(LELM(AA))+ WEIGHT(AA)) .EQ. DIST(V)) GOTO 130
         CALL ERROR(6HSHPTHW)
  130    INSERT AA INTO SHPTHW
         V = LELM(AA)
         GOTO 110
  300    RETURN
         END
```

The aim of solving this problem was to demonstrate the use of the ALLA language and typical types of interaction. There is nothing special about the method used for the computation. It is already a commonly known algorithm of E. F. Moore.[44] It begins by assigning an associated distance of infinity (actually ten million here) to each vertex in the given graph. The starting vertex is given a distance of 0. The distance property represents the minimum known path cost from the

starting vertex. Initially, the distance assigned to the starting vertex is trivially known to be 0. All other distances are assumed to be infinite since without considering the graph's connectivity all other vertices are potentially unreachable.

The SHPTHW function assumes each vertex of the given graph has a set of INARC's and a set of OUTARC's (see Section 3.19.3). The algorithm consists of two parts: a search followed by a trace. The search begins at the starting vertex and a distance is assigned to each vertex which can be reached from the starting vertex by traversing one outgoing arc. Next, distances are assigned to vertices which are connected by an outgoing arc to the vertices just assigned. The distance of a next vertex is equal to the distance of the previous vertex plus the weight of the arc. If a distance had already been assigned to a vertex, it is replaced with a new distance only when the new distance is numerically smaller. This process is repeated until a pass is made which does not improve any distance in the graph. At this time, the distance associated with each vertex is the minimum cost to reach that vertex starting from the given starting vertex.

In terms of the SHPTHW function, TDSET is the set of previously assigned vertices as each pass occurs. DSET is the set used to keep track of the new vertices being assigned during a pass. The search part of the algorithm is between statements 20 through two statements past statement 40. Then there is a check for whether the given ending vertex has been assigned a distance. If its distance has remained infinite, then it cannot be reached and there is no trace. Otherwise, the trace begins near statement 110.

The trace starts with the ending vertex. Each incoming arc is considered along with the vertex from which the arc emanates. If the distance of that vertex plus the weight of that arc equals the distance assigned to the ending vertex, that arc is part of a shortest path. This process continues until the starting vertex is reached at which point the answer has been computed.

The SHPTHW described above performs the necessary computation to find the shortest path through a graph, but in order to interface to a user, an interactive ALLA routine must be used. The subroutine SHPATH has been written for this purpose. A source listing follows.

```
      SUBROUTINE SHPATH
      ENTITY G,A,FROMV,TOV,SHP,SHPTHW
      INTEGER WEIGHT,LABEL1,INNAME,GMISC,BITS,ASCDEC
      LOGICAL SWITCH
      PROPERTY WEIGHT
  5   DOG BLINKM
      GETGRAPH G
      CALL INOUT(G)
      THROUGH 10 FORALL A IN RELM(G)
      WEIGHT(A) = ASCDEC(LABEL1(A))
 10   REMPROP LABEL1 FROM A
 20   SWITCH = .FALSE.
      DOG START LTPEN WHOLE VERTEX,ALL
      DOG STOP LTPEN WHOLE VERTEX LIST
      THROUGH 25 FORALL FROMV IN LELM(G)
      IF (BITS(33,35,GMISC(FROMV)) .NE. 1) GOTO 25
      DOG (INNAME(FROMV))
 25   CONTINUE
      DOG 0
      CALL MESSAG(3)
      DOGSTRING 'SELECT A STARTING VERTEX'
      CALL SELECT
      THROUGH 30 FORALL FROMV IN LELM(G)
 30   IF (INNAME(FROMV) .EQ. LPHIT2) GOTO 40
 35   CALL MESSAG(3)
      DOGSTRING 'NO VERTEX SELECTED'
      SWITCH = .TRUE.
      GOTO 90
 40   DOG START EXIST SHAPE VERTEX 4,(LPHIT2)
      DOG STOP BLINK WHOLE VERTEX,(LPHIT2)
      DOG STOP LTPEN WHOLE VERTEX,(LPHIT2)
      CALL MESSAG(3)
      DOGSTRING 'SELECT ENDING VERTEX'
      CALL SELECT
      THROUGH 50 FORALL TOV IN LELM(G)
 50   IF (INNAME(TOV) .EQ. LPHIT2) GOTO 60
      DOG START EXIST SHAPE VERTEX 2,(INNAME(FROMV))
      GOTO 35
```

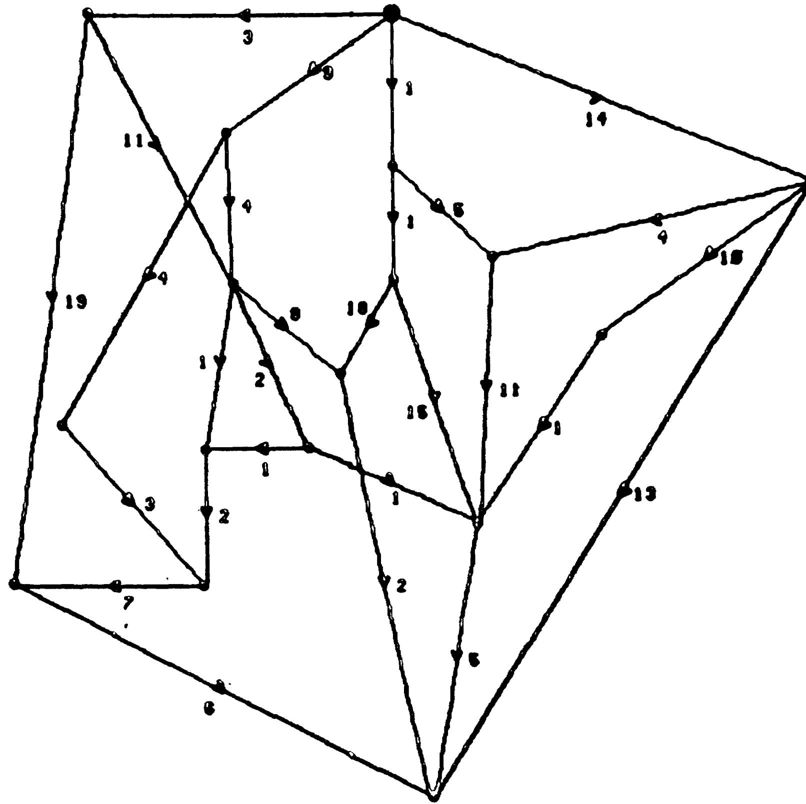
```
60      DOG START EXIST SHAPE VERTEX 7,(LPHIT2)
        DOG STOP BLINK WHOLE VERTEX,(LPHIT2)
        SHP = SHPTHW(FROMV,TOV,G)
        IF (EMPTY(SHP)) GOTO 80
        DOG DIMM 3
        DOG START DIM WHOLE ARC 7 LIST
        THROUGH 70 FORALL A IN SHP
70      DOG (INNAME(A))
        DOG 0
        DELETE SHP
        CALL MESSAG(3)
        DOGSTRING 'A SHORTEST PATH IS SHOWN'
        GOTO 100
*
80      CALL MESSAG(3)
        DOGSTRING 'THERE IS NO PATH'
90      DOG DIMM 5
100     CALL MESSAG(2)
        DOGSTRING 'PB 10 FOR ANOTHER PAIR'
        CALL MESSAG(1)
        DOGSTRING 'PB 11 TO ALTER GRAPH'
        DOG 0
        DOG START DIM WHOLE VERTEX LIST+INTENS,4093,4094,4095,0
        CALL CLRPB(10,11)
200     WAITCHANGE
        IF (.NOT.(PB(10).OR.PB(11))) GOTO 200
        IF (SWITCH) GOTO 210
        DOG START EXIST SHAPE VERTEX 2 LIST
        DOG (INNAME(FROMV)), (INNAME(TOV)), 0
210     IF (PB(10)) GOTO 20
        DOG BLINKM
        CALL DELGRA(G)
        ESCAPE
        GOTO 5
        END
```

The SHPATH subroutine begins by getting the current graph from the DEC-338 and computing incoming and outgoing arcs. Next, each arc label is replaced by an integer representing its weight. The user is then given a choice of selecting a starting vertex using any vertex of the graph whose shape is not 1 (used for bends in bent arcs). It is assumed that all vertices (which are not bends) are initially of shape 2. When the user selects a starting vertex, it is changed to shape 4, which makes it a larger dot. The user is then requested to choose an ending vertex. When this is done, it is changed to shape 7, a larger circular shape. The next step calls upon the SHPTHW function to compute a shortest path. If the answer is an empty set the user is informed that there is no path. Otherwise, the whole graph is dimmed down to intensity level 3, and the arcs comprising the path are intensified to level 7. In order to remain conspicuous the user message lines are brightened back to the overall picture intensity level. The user is given a choice of either choosing another pair of vertices or altering the graph. The latter choice causes the old graph to be deleted and control is given to the local manipulation facilities of the Graph Monitor. Later, when the user resumes execution the subroutine restarts by getting the graph again.

Figure 6-1 shows a weighted graph after a user has selected a starting vertex. Figure 6-2 shows the result of a shortest path computation.

6.3 Tree Layout

This example better demonstrates the advantages of an interactive system since a problem is attacked which is concerned with human judgment. The user is expected to call upon this routine with a special



SELECT ENDING VERTEX
OR PB 10 FOR NO SELECTION

Figure 6-1 A Weighted Graph

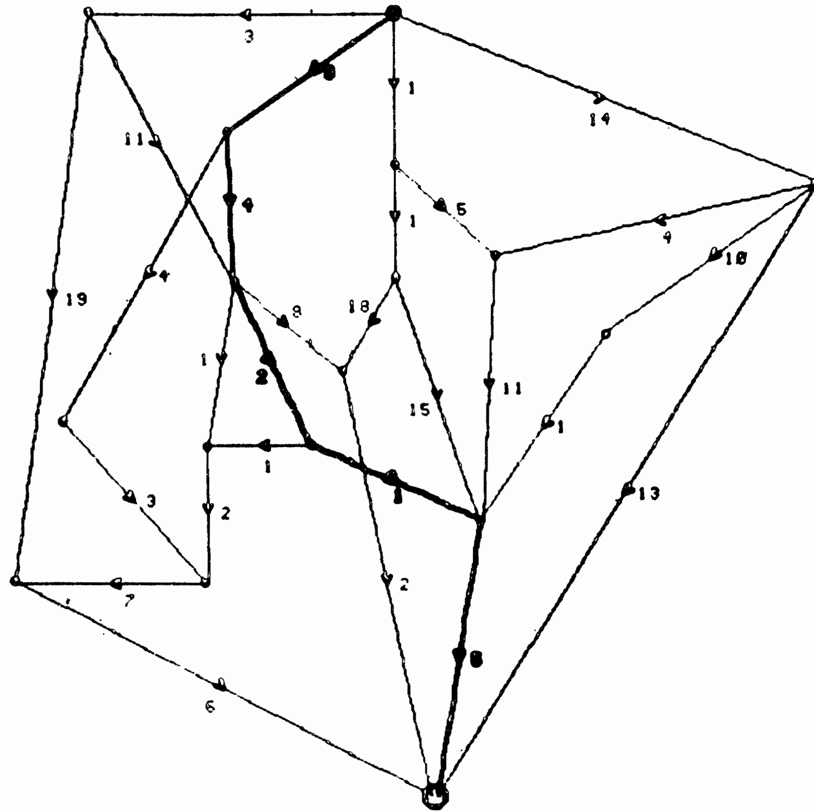


Figure 6-2 A Shortest Path Computed

type of graph displayed at the terminal. The type of graph on which the algorithm operates is called a tree. When regarded as undirected this type of graph is one which has no cycles. This means there are no paths in the graph which have the same starting vertex and ending vertex. An equivalent definition of a tree is: a graph whose number of vertices is exactly one more than its number of arcs. If a graph is regarded as directed, then a definition of a tree becomes more involved. It must have a unique vertex called a root, and starting from the root, each vertex must be reachable by exactly one path. Section 3.19.5 gave an algorithm for determining whether a given graph is an undirected tree in the form of a logical function named NDTREE. Section 3.19.6 presented the logical function DTREE used for determining whether a given graph is a directed tree with a particular vertex as a root. Both of these functions are used within the subroutines written to perform tree layout.

The example which has been implemented accepts an undirected tree and a user-selected vertex to be taken as the root. Any arrows of the given graph are then re-oriented so the graph is a directed tree with the chosen vertex as root. The computer then repositions the root at the top of the screen at the center. Each vertex of depth 1 (namely, those which are at the other end of all arcs leaving the root) are repositioned along the same horizontal line below the root. All vertices of depth 2 are repositioned along a lower horizontal line, etc. This is done in such a way that no arc crosses another. Also, the relative positions of vertices in the given graph are preserved as long as the constraint of tree-like layout can be met. The tree is positioned to nearly fill the display screen, independently of window size.

After a tree layout has occurred the user is given three choices: he may retain the graph and permute arcs, he may select another root, or he may revert to the Graph Monitor for arbitrary graph alteration. The first alternative is used to move a vertex to a new position on the horizontal relative to other vertices of the same depth. All arcs and vertices below the repositioned vertex will retain their relative positions.

The organization of the program which underlies this example is of the same form as that used in the shortest path example. A subroutine named TREELA is the main program which includes all of the interactive operations. It calls upon the subroutine LAYOTR which performs the tree layout algorithm. Although the subroutine is called in a fixed way, it is written to perform a tree layout in a given rectangular area according to four integer arguments representing lower and upper x- and y-coordinates.

In view of the previous description, the source listing of the TREELA subroutine follows without further explanation.

```
      SUBROUTINE TREELA
      ENTITY G,ROOT,V,V1,LIST,A,INARC,OUTARC,NEXTV
      INTEGER INNAME,YCOORD,XCOORD
      LOGICAL NOTREE
      PROPERTY INARC,OUTARC,XCOORD
 10   GETGRAPH G
      IF (NOTREE(G)) GOTO 30
      CALL MESSAG(2)
      DOGSTRING 'THIS IS NOT A TREE'
 15   CALL MESSAG(1)
      DOGSTRING 'PB 11 TO ALTER GRAPH'
      CALL CLRPB(11)
 20   WAITCHANGE
      IF (.NOT.PB(11)) GOTO 20
 25   CALL DELGRA(G)
      ESCAPE
      GOTO 10
 30   DOG START LTPEN WHOLE VERTEX, ALL
      CALL MESSAG(3)
      DOGSTRING 'SELECT A ROOT'
      CALL SELECT
      THROUGH 40 FORALL ROOT IN LELM(G)
 40   IF (INNAME(ROOT) .EQ. LPHIT2) GOTO 50
      CALL MESSAG(3)
      DOGSTRING 'NO VERTEX SELECTED'
      GOTO 15
 50   CALL INOUT(G)
      CALL UNDIR(G)
      DOG START EXIST SHAPE VERTEX 7,(LPHIT2)
      DOG STOP BLINK WHOLE VERTEX,(LPHIT2)
      INSERT ROOT INTO CRSET(LIST)
      THROUGH 80 FORALL V IN LIST
      THROUGH 70 FORALL A IN OUTARC(V)
      IF (LELM(A) .EQ. V) GOTO 60
      RELM(A) = LELM(A)
      LELM(A) = V
      DOG START EXIST NAMES ARC,(INNAME(A))
      DOG  (INNAME(RELM(A))), (INNAME(LELM(A)))
 60   NEXTV = RELM(A)
      REMOVE A FROM OUTARC(NEXTV)
      INARC(NEXTV) = CRSET(TEMP)
      INSERT A INTO TEMP
 70   INSERT NEXTV ONTO LIST
 80   REMOVE V FROM LIST
      DOGFLUSH
      CALL MESSAG(1)
      DOGSTRING 'COMPUTING'
      DOGFLUSH
```

```
   90    CALL LAYOTR (G,ROOT,1124,1948,1224,1948)
         DOG START EXIST COORDS VERTEX LIST
         THROUGH 100 FORALL V IN LELM(G)
  100    DOG (INNAME(V)), (YCOORD(V)), (XCOORD(V))
         DOG 0
         DOG START EXIST SHAPE VERTEX 2,(LPHIT2)
  150    CALL MESSAG(3)
         DOGSTRING 'PB 9 TO PERMUTE ARCS'
         CALL MESSAG(2)
         DOGSTRING 'PB 10 FOR ANOTHER ROOT'
         CALL MESSAG(1)
         DOGSTRING 'PB 11 TO ALTER GRAPH'
         CALL CLRPB(9,10,11)
  200    WAITCHANGE
         IF (PB(9)) GOTO 290
         IF (PB(10)) GOTO 30
         IF (PB(11)) GOTO 25
         GOTO 200
  290    CALL MESSAG(3)
         DOGSTRING 'POINT TO A VERTEX TO MOVE'
         DOG STOP LTPEN WHOLE VERTEX, (INNAME(ROOT))
         CALL SELECT
         DOG START LTPEN WHOLE VERTEX, (INNAME(ROOT))
         THROUGH 291 FORALL V1 IN LELM(G)
  291    IF (INNAME(V1) .EQ. LPHIT2) GOTO 292
         GOTO 150
  292    DOG PSEUDO, (LPHIT3), (LPHIT4)
         DOG SETCUR, PENPNT 924, PENPNT
         CALL CLRPB(7,11)
         CALL SETPB(8)
         DOG START CURSOR
         CALL MESSAG(3)
         DOGSTRING 'MOVE PEN TO NEW RELATIVE'
         CALL MESSAG(2)
         DOGSTRING ' POSITION OF THE VERTEX'
         CALL MESSAG(1)
         DOGSTRING 'PB 11 WHEN DONE'
  293    WAITCHANGE
         IF (.NOT. PB(11)) GOTO 293
         DOG STOP CURSOR
         CALL CLRPB(8)
         DOG STOP BLINK WHOLE VERTEX, (LPHIT2)
         IF (CHPSEU .EQ. 0) GOTO 150
         XCOORD(V1) = 1024 + XPSEUD
         GOTO 90
         END
```

The LAYOTR subroutine is the heart of the tree layout process. Its arguments are the graph (tree) and its root plus four arguments signifying the rectangular area to be occupied. The purpose of the subroutine is to assign new XCOORD and YCOORD properties to each vertex of the given graph. The YCOORD is assigned only according to the depth of a particular vertex and the maximum depth in the given tree. In order to assign values to x-coordinates, the OUTARC set for each vertex is first ordered according to the x-coordinates in the given tree. Then a pass is made for each level of the tree until all endpoints are collected into an ordered set according to the ordering of the OUTARC sets. Endpoints are then assigned x-coordinates so that they are spread equidistantly. This assignment determines the rest of the positioning since each remaining vertex is assigned an x-coordinate which is the average value of the x-coordinates of those vertices which are connected directly below it. This part of the ALLA program demonstrates the use of a pushdown list.

A source listing of the LAYOTR subroutine follows:

```
          SUBROUTINE LAYOTR(G,ROOT,XXMIN,XXMAX,YYMIN,YYMAX)
          ENTITY G,ROOT,ENDP,V,PDL,CURRV,OUTARC
          ENTITY NEWOUT,OUTAV,A,MINARC,NEWEND
          INTEGER XXMIN,XXMAX,YYMIN,YYMAX,XMIN,XMAX,YMIN,YMAX
          INTEGER TEMP,D,SIZE,SUM,N,YCOORD,XCOORD,DEPTH,CARD
          INTEGER XC,MINX
          REAL NUM,DEN,DELTA
          LOGICAL DTREE,SWITCH
          PROPERTY YCOORD,XCOORD,OUTARC
          IF (DTREE(G,ROOT,TEMP)) GOTO 10
          CALL ERROR(25H'LAYOTR' GIVEN A NON-TREE)
   10     XMIN = MINO(IABS(XXMIN),IABS(XXMAX))
          XMAX = MAXO(IABS(XXMIN),IABS(XXMAX))
          YMIN = MINO(IABS(YYMIN),IABS(YYMAX))
          YMAX = MAXO(IABS(YYMIN),IABS(YYMAX))
          THROUGH 80 FORALL V IN LELM(G)
          CREATE SET NEWOUT
          OUTAV = OUTARC(V)
   50     IF (EMPTY(OUTAV)) GOTO 70
          MINX = 1000000
          THROUGH 60 FORALL A IN OUTAV
          XC = XCOORD(RELM(A))
          IF (XC .GE. MINX) GOTO 60
          MINARC = A
          MINX = XC
   60     CONTINUE
          INSERT MINARC INTO NEWOUT
          REMOVE MINARC FROM OUTAV
          GOTO 50
   70     DELETE OUTAV
   80     OUTARC(V) = NEWOUT
          INSERT ROOT INTO CRSET(ENDP)
          D = 0
  100     D = D + 1
          SWITCH = .FALSE.
          CREATE SET NEWEND
          THROUGH 130 FORALL V IN ENDP
          OUTAV = OUTARC(V)
          IF (EMPTY(OUTAV)) GOTO 120
          SWITCH = .TRUE.
          THROUGH 110 FORALL A IN OUTAV
  110     INSERT RELM(A) INTO NEWEND
          GOTO 130
  120     INSERT V INTO NEWEND
  130     CONTINUE
```

```
          DELETE ENDP
          ENDP = NEWEND
          IF (SWITCH) GOTO 100
          NUM = YMAX - YMIN
          DEN = D - 1
          IF (D.NE.1) DELTA = NUM/DEN
          THROUGH 150 FORALL V IN LELM(G)
          XCOORD(V) = -1
          NUM = DEPTH(V) - 1
          SIZE = NUM*DELTA
    150   YCOORD(V) = YMAX - SIZE
          NUM = XMAX - XMIN
          DEN = CARD(ENDP) - 1
          IF (DEN.NE.O.) DELTA = NUM/DEN
          NUM = XMIN
          THROUGH 160 FORALL V IN ENDP
          SIZE = NUM
          XCOORD(V) = SIZE
    160   NUM = NUM + DELTA
          DELETE ENDP
          PUSH ROOT ONTO CRSET(PDL)
    200   IF (EMPTY(PDL)) GOTO 300
          CURRV = POP(PDL)
          SUM = 0
          N = 0
          SWITCH = .FALSE.
          THROUGH 230 FORALL A IN OUTARC(CURRV)
          SIZE = XCOORD(RELM(A))
          IF (SIZE.NE.(-1)) GOTO 220
          IF (SWITCH) GOTO 210
          SWITCH = .TRUE.
          PUSH CURRV ONTO PDL
    210   PUSH RELM(A) ONTO PDL
          GOTO 230
    220   IF (SWITCH) GOTO 230
          SUM = SUM + SIZE
          N = N + 1
    230   CONTINUE
          IF (SWITCH) GOTO 200
          XCOORD(CURRV) = SUM/N
          GOTO 200
    300   DELETE PDL
          RETURN
          END
```

Figure 6-3 shows a graph which is a tree as it might be created by a user. After applying the tree layout function, the vertices have been repositioned to yield Figure 6-4. Next, the "permute arcs" option was taken and the tree was further changed to Figure 6-5. As one more example of use, another root was then selected which resulted in another layout process as shown in Figure 6-6.

6.4 Maximally Complete Subgraphs

A problem which arises in the organization of information, such as automatic classification and automatic indexing, is "clustering" or "clumping" of descriptors. This problem has been posed with a graph theoretic model where the descriptors are represented by vertices. A relationship between two descriptors is represented by an undirected arc joining the two corresponding vertices. The problem of finding clusters is represented by the determination of maximally complete subgraphs. [5,69,74]

A complete subgraph of a given graph is composed of some subset of the vertices of a graph where there is an arc connecting each pair of vertices in the subset. A maximally complete subgraph (MCS) of a given graph is a complete subgraph of a given graph which is not a subgraph of any other complete subgraph of the given graph.

Previous work carried out by the author resulted in a report which presented algorithms for the determination of all maximally complete subgraphs of a given graph.[74] The following formulation of the "Basic MCS Algorithm" is taken from the flowchart on page 17 of that report.

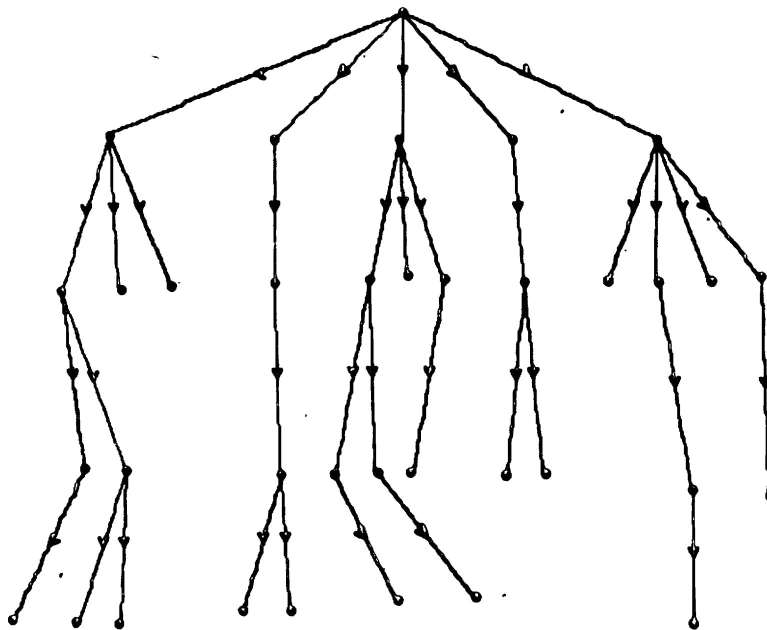
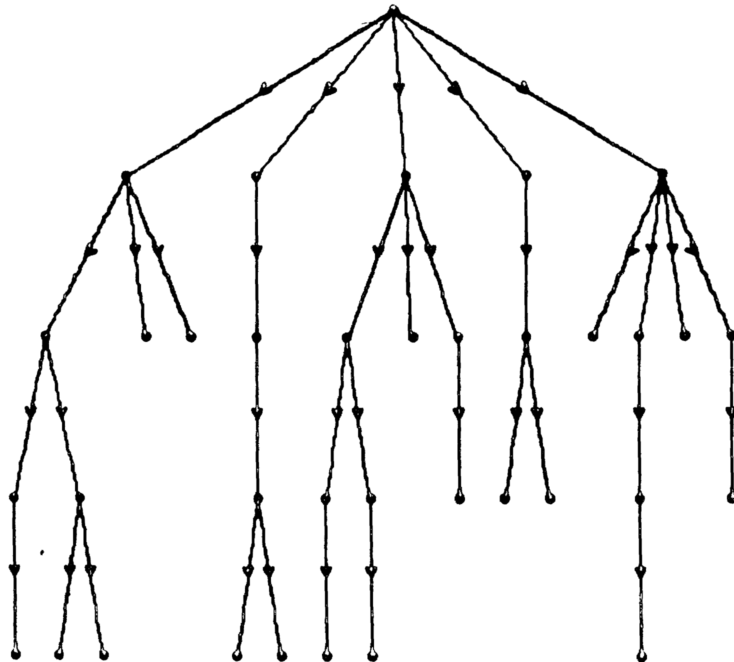


Figure 6-3 A Tree



PB 9 TO PERMUTE ARCS
PB 10 FOR ANOTHER ROOT
PB 11 TO ALTER GRAPH

Figure 6-4 The Same Tree After Layout

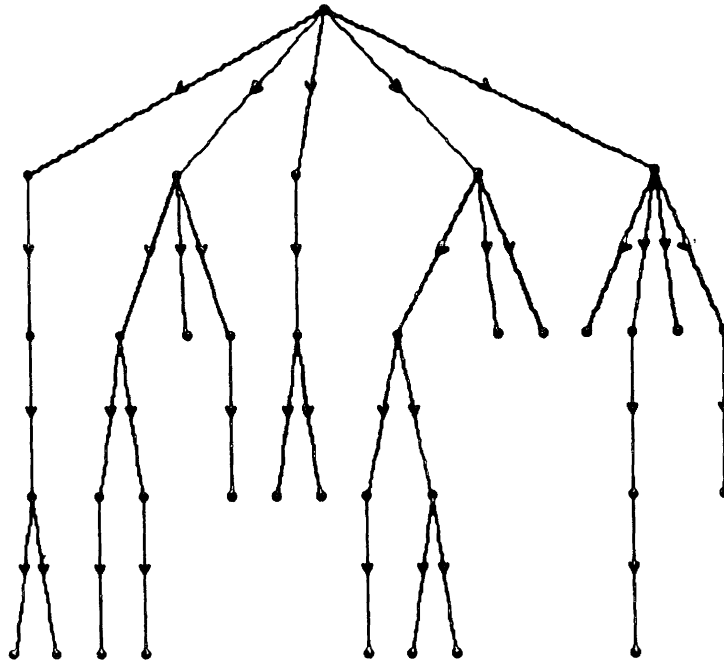
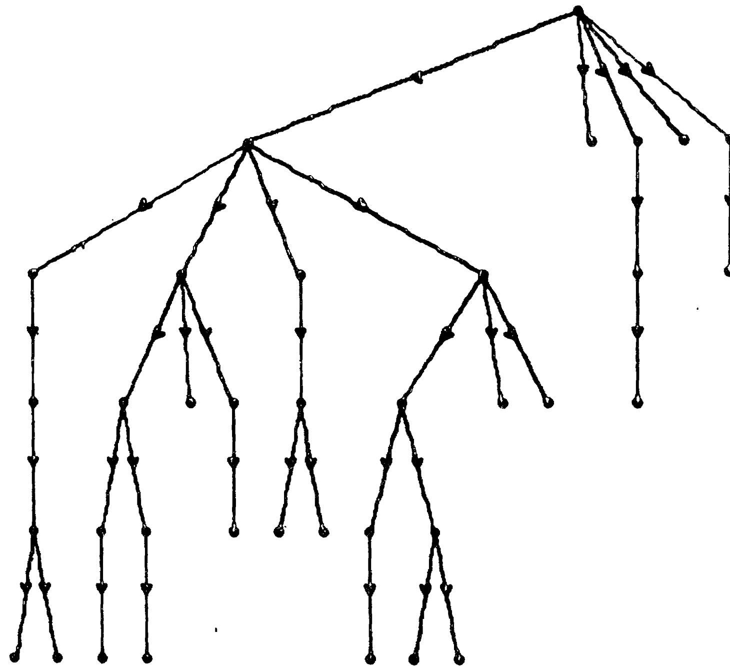


Figure 6-5 The Same Tree with Arcs Permuted



PB 9 TO PERMUTE ARCS
PB 10 FOR ANOTHER ROOT
PB 11 TO ALTER GRAPH

Figure 6-6 The Same Tree with Another Root

Step 1: $1,2,\ldots,N \rightarrow Y[0]$
$0 \rightarrow \ell(0)$
$0 \rightarrow k$

Step 2: $\ell(k) \rightarrow \ell(k+1)$

Step 3: $Y[k]/\ell(k+1) \rightarrow \ell(k+1)$
and go to step 7 if null

Step 4: $k + 1 \rightarrow k$
$Y[k-1] \cap X[\ell(k)] \rightarrow Y[k]$

Step 5: If Y[k] is not empty, go to step 2

Step 6: Output MCS: $\ell(1),\ell(2),\ldots,\ell(k)$

Step 7: If $k = 0$ terminate

Step 8: $k - 1 \rightarrow k$
and to to step 3

The above formulation of the algorithm operates on integers which represent vertices. The graph has N vertices which are arbitrarily assigned unique integers: $1,2,\ldots,N$. Associated with each vertex j is a set X[j] of vertices which are the immediate neighbors of that vertex. MCS's are built up by adding one vertex at a time to a candidate list of those vertices being considered as a possible MCS. This list is named $\underline{\ell}$, and the variable $\underline{k}$ keeps track of the number of members of the $\underline{\ell}$ list. The algorithm uses a list of sets of vertices: Y[1],Y[2],..., where each Y[j] is the ordered set of those vertices which are connected to vertices $\ell(1),\ell(2),\ldots,$ and $\ell(j)$.

The special notation of step 3 above indicates that $\ell(k+1)$ is replaced by the smallest member of Y[k] which is greater than $\ell(k+1)$.

Starting with the above formulation of the algorithm each line was encoded into an equivalent ALLA statement. Since FORTRAN subscripts begin with 1, the range of the variable $\underline{k}$ has been increased by one. The ALLA version is an improvement over the above formulation since it treats vertices as abstract entities rather than integers. This change necessitates another definition of the operation performed in step 3 above. Instead of numerical ordering, the algorithm picks the next vertex according to the original arbitrary order of the set of vertices of the graph. The subroutine which creates the set of neighbors preserves the ordering. Since the function for set intersection also preserves the ordering, the ordering of the Y sets reflects the original ordering of vertices.

The following listing of the MCS1 subroutine should be compared with the above eight steps. Step 6 has been expanded to ten card images, and the termination portion of the subroutine begins at statement 40. The original algorithm is otherwise found in the listing as a line-by-line translation.

At statement 5, a subroutine call is specified which creates the X lists (here named NEIGH). At statement 20, the NEXT function finds the next vertex in the given set as performed in step 3 above. At three statements after statement 20, the INTERS function is applied, which computes the set intersection of two given sets. Listings of CRNEIG, NEXT, and INTERS follow the listing of MCS1.

```
      SUBROUTINE MCS1
      ENTITY L(40),Y(40),G,INTERS,NEIGH,NEXT
      INTEGER K,I,INNAME
5     CALL CRNEIG(C)
      Y(1) = LELM(G)
      L(1) = UNDEF
      K = 1
10    L(K+1) = L(K)
20    L(K+1) = NEXT(Y(K), L(K+1))
      IF (NULL(L(K+1))) GOTO 30
      K = K + 1
      Y(K) = INTERS(Y(K-1), NEIGH(L(K)))
      IF (.NOT. EMPTY(Y(K))) GOTO 10
      DOG START EXIST SHAPE VERTEX 2, ALL
      CALL MESSAG(2)
      DOGSTRING 'AN MCS IS SHOWN'
      CALL MESSAG(1)
      DOGSTRING 'PB 4 TO SEE NEXT ONE'
      CALL SETPB(3)
      DOG START EXIST SHAPE VERTEX 6 LIST
      DO 25 I = 2,K
25    DOG (INNAME(L(I)))
      DOG 0,0
30    IF (K .EQ. 1) GOTO 40
      DELETE Y(K)
      K = K - 1
      GOTO 20
*
40    DOG START EXIST SHAPE VERTEX 2, ALL
      CALL MESSAG(2)
      DOGSTRING *NO MORE MCS'S*
      CALL MESSAG(1)
      DOGSTRING 'PB 11 TO ALTER GRAPH'
      CALL CLRPB(3,11)
50    WAITCHANGE
      IF (PB(11)) GOTO 60
      GOTO 50
60    CALL DELGRA(G)
      ESCAPE
      GOTO 5
      END
```

```
      SUBROUTINE CRNEIG(G)
      ENTITY G,V1,V2,A,X,NEIGH,OUTARC
      INTEGER DEPTH,I
      PROPERTY NEIGH,DEPTH,OUTARC
      GETGRAPH G
      CALL INOUT(G)
      CALL UNDIR(G)
      I = 1
      THROUGH 40 FORALL V1 IN LELM(G)
      NEIGH(V1) = CRSET(X)
      DEPTH(V1) = I
      I = I + 1
      THROUGH 30 FORALL V2 IN LELM(G)
      IF (V1 .EQ. V2) GOTO 30
      THROUGH 10 FORALL A IN OUTARC(V1)
      IF (LELM(A) .EQ. V2) GOTO 20
   10 IF (RELM(A) .EQ. V2) GOTO 20
      GOTO 30
   20 INSERT V2 INTO X
   30 CONTINUE
      DELETE OUTARC(V1)
      REMPROP OUTARC FROM V1
   40 CONTINUE
      RETURN
      END
```

```
      ENTITY FUNCTION NEXT(S,E)
      ENTITY S,E,X
      INTEGER D,DEPTH
      D = DEPTH(E)
      THROUGH 10 FORALL X IN S
   10 IF (DEPTH(X) .GT. D) GOTO 20
      X = UNDEF
   20 NEXT = X
      RETURN
      END
```

```
      ENTITY FUNCTION INTERS(A,B)
      ENTITY A,B,E
      CREATE SET INTERS
      THROUGH 10 FORALL E IN A
      IF (.NOT. MEMBER(E,B)) GOTO 10
      INSERT E INTO INTERS
   10 CONTINUE
      RETURN
      END
```

Figure 6-7 shows an undirected graph drawn at the DEC-338 as a copy of a figure in the report "Determination of Maximally Complete Subgraphs" (page 9).[74] The Basic MCS algorithm was applied to the graph. Since all vertices of an MCS are mutually connected, it suffices to cite each MCS as its set of vertices. The graph of Figure 6-7 has vertices labelled with integers, but the algorithm is independent of such labels. Its output is in terms of the actual vertices of the graph. Figure 6-8 shows a first MCS computed by making square those vertices forming the MCS: 1, 3, 5, 9, 11, 12. The user then depresses pushbutton 4 to see the next one, and immediately the display screen appears as in Figure 6-9.

A feature of interactive execution which makes this speed possible is the single-step option. When pushbutton 3 is ON, and the interactive ALLA program has generated a DOGGIE command word of 0000, the user program running in the DEC-338 stalls when that word is detected until pushbutton 4 is depressed.

Figure 6-10 shows a third MCS computed as it appears when the user again depresses pushbutton 4. As the user continues to request MCS's, they immediately appear on the screen until all have been shown. In this example, the remaining MCS's of the given graph are:

2,9,11
3,5,7,8,12
3,8,10
3.9,10
4,6
4,9,11

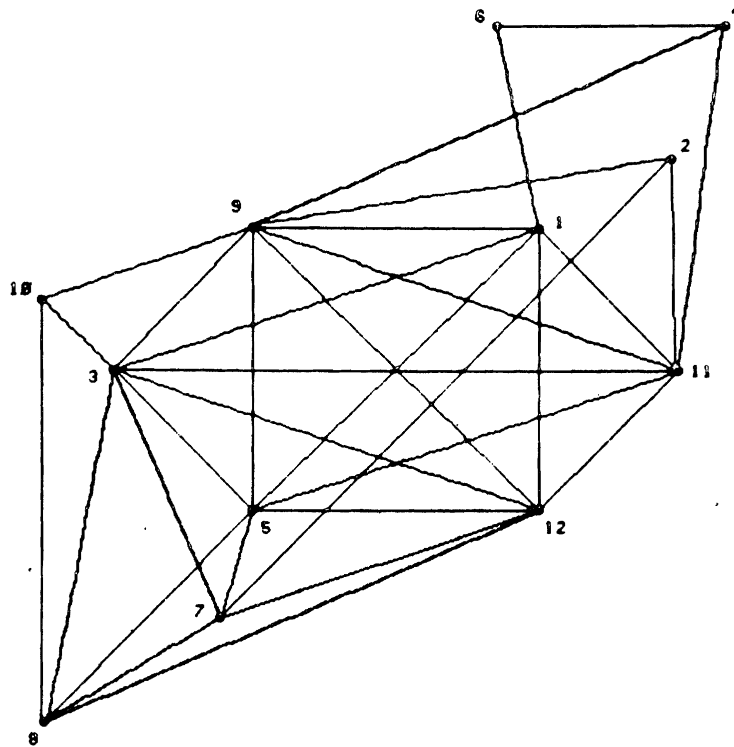


Figure 6-7 An Undirected Graph

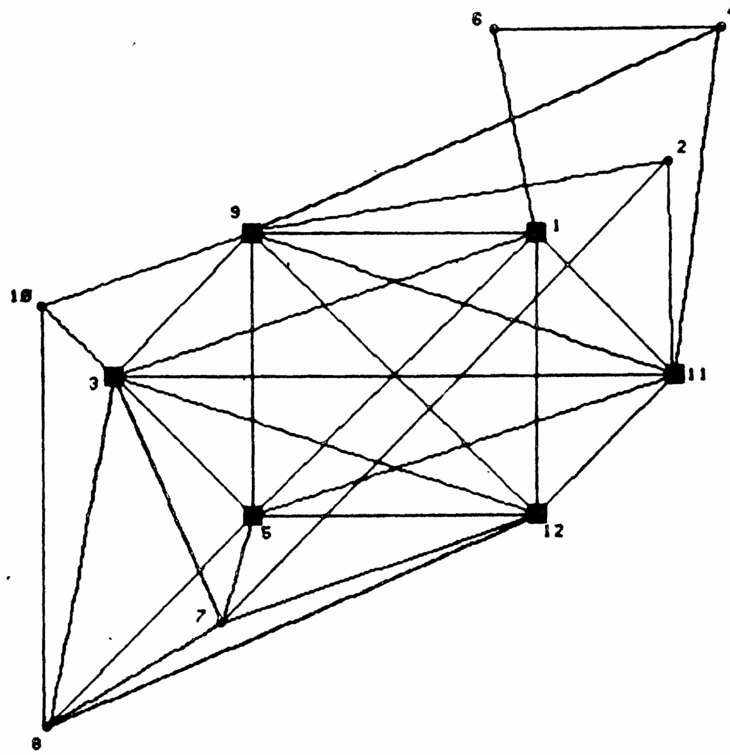


Figure 6-8 An MCS Computed

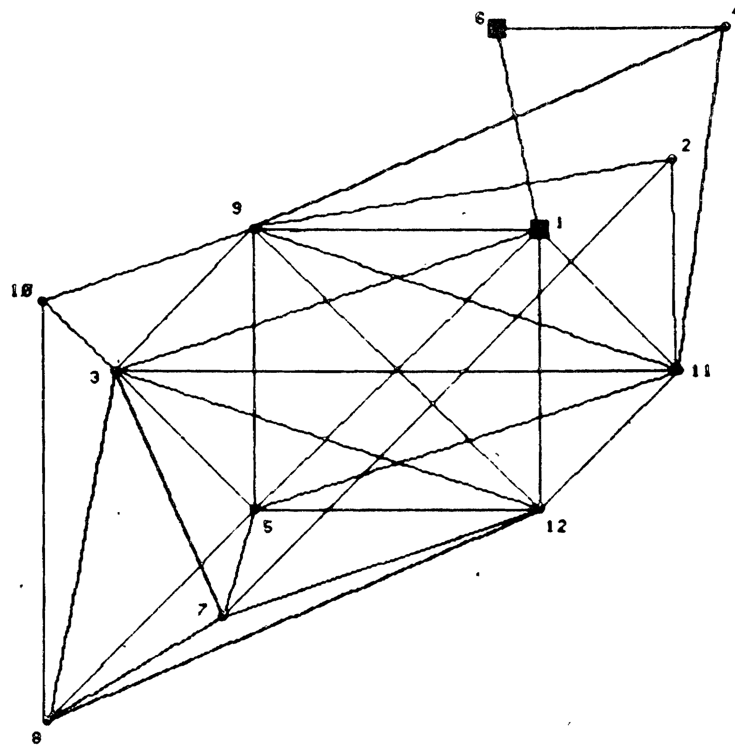


Figure 6-9 Another MCS Computed

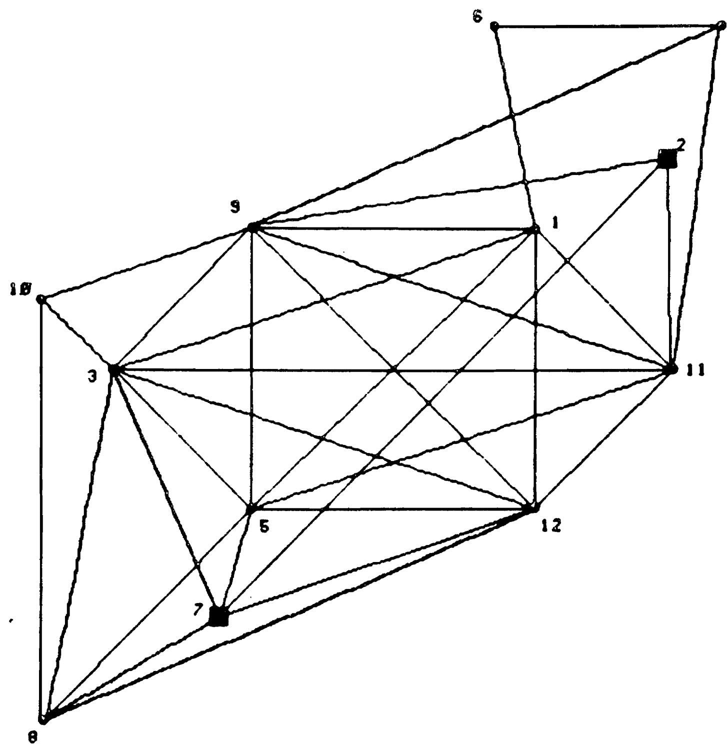


Figure 6-10 A Third MCS Computed

CHAPTER 7

CONCLUSIONS

This dissertation has described how a remote computer graphics terminal was connected into an existing multi-console operating system which previously supported only Teletype and alphanumeric display terminals. In this environment an Interactive Graph Theory System was built where modularity and flexibility have been stressed. The DEC-338 is used both as an alphanumeric console and a graphics terminal. The design has departed from other efforts involving a computer graphics terminal with processing power. In this system the terminal may be programmed to perform both local functions and functions which are performed as subroutines of the interactive program running in the central computer.

The FORTRAN IV language was enriched with data structure and associative operations and also interactive components, yielding the interactive ALLA language. This language includes those set-theoretic operators which permit the handling of graphs. Chapters 3 and 6 have demonstrated the effectiveness of ALLA by showing its readability and power of expression.

A valuable part of the system is the DOGGIE executive and interpreter which resides in the DEC-338. The use of the DOGGIE command language in both the central and terminal computers has been instrumental in the ease and speed with which the system was developed. It has unified the system organization, which is reflected in a unified description in Chapter 4.

A library of interactive algorithms has been started. The system is ready to be used by a programmer who wishes to pursue a particular application. Before a non-programmer may benefit, however, the system must be extended with more graph-theoretic algorithms. These would include both generators and recognizers of certain types of graphs such as symmetric graphs, regular graphs, and complete graphs. Functions of one or more graphs would include computation of cyclomatic number, chromatic number, a graph coloring, internal stability, minimum cover, homomorphism, etc. The generation of a random graph is another useful function which should be available. There are commonly needed functions for graph manipulations such as automatic labeling; duplicating, moving, or deleting subgraphs; and some general methods of selecting a subgraph or set of entities. The tree layout algorithm might be extended to help layout lattices or perhaps graphs of arbitrary connectivity.

The current method of initiating interactive execution is oriented towards the programmer's view of the system. A user program could be written to provide the user with a menu of available functions. In order to improve initial connection time, a user program could be written to automatically sign in (or log in) a user of the Interactive Graph Theory System.

The use of the DEC-338 in this system has demonstrated the limits of an 8K-word machine with a 32K minidisk. The available space limits graphs to be no larger than about 60 labeled vertices and arcs, but flicker of the display screen starts to become a problem at this point. There is no room to save graphs on the minidisk. A future project could consist of saving and restoring graphs on DECtape. This could

remove the burden from the central computer. However, the central computer saves graphs by description and the power of the MULTILIST facilities in dealing with graphs is often more significant.

In the design of DOGGIE, there has been no special treatment of user messages. Instead, messages are labels of vertices of null shape (or shape 3 for light buttons). This has been a disadvantage since messages are the most commonly created objects. If the notion of a message had been embodied in the DOGGIE command language, the coding of the language would be more compact, but generality now present would be lost. The DOGGIE program itself would also have to be larger. Another disadvantage of the way messages are now handled is the DIMM command and the ALL option must refer to the messages as well as the graph. However, the control which is now available has been used, and therefore it would be too severe a limitation to permit a change.

In the IBM 7040, the use of L^6 to implement the underlying memory structure has proven advantageous. This was especially important during the development of the memory structure, since it could be easily modified as program bugs were detected. For speed of processing, the memory structure routines were coded without error diagnostics, but this has forced the author to resort to either L^6 or IBSYS dumps during the debugging of some of the ALLA programs. Future work could include the expansion of the memory structure routines into an alternate package to satisfy this deficiency. As emphasized previously, this type of system modification may be carried out from the remote terminal. Another topic for future work is the extension or modification of the memory structure package either to create more primitives such as set operators or to represent relationships differently such as employing uni-direc-

tional linkage.

The Interactive Graph Theory System presently employs one data structure and memory structure. Certain types of problems may have significantly simpler solutions when posed under an alternate organization of the data. A valuable extension of the current system would permit at least a variety of memory structures, and also possibly data structures. Perhaps a programmer would direct the conversions among the structures, or such transformations might occur automatically. This approach then leads to the problem of transforming algorithms consistently with the data transformations.

The existing system may now be used to solve a variety of individual problems. Packages dedicated to particular applications can be developed which aid in teaching graph theory and finite state machines. The interactive layout techniques may be used for presentation or publication. Each reader of this dissertation has probably developed his own ideas of what this system has to offer and how it may be developed to encompass further applications.

APPENDIX 1

INTERNAL ORGANIZATION OF DOGGIE

A1.1 Introduction

DOGGIE (for Display of Graphs Graphical Interpretive Executive) is the program resident in the DEC-338 during any graphical manipulation in the Interactive Graph Theory System. The heart of DOGGIE is the interpreter which decodes and executes strings of 12-bit words. This appendix describes DOGGIE at a level which a user or programmer of the Interactive Graph Theory System need not know. It is written for the interested person or for the maintainer of the program. Knowledge of the PDP-8, DEC-338, and PDP-8 Disk Monitor System is assumed.[12,53,54,73]

DOGGIE is written in PDPMAP Assembly Language and consists of two source decks, named GPACK and CHTBL. The latter is only the dispatch table and increment-mode subroutines for alphanumeric character shapes. Since the assembly listings of GPACK and CHTBL consist of more than 100 computer printout pages, they are not included in this report. The listings, however, are very fully commented and serve as the best explanation of the program's operation at the detailed level. This appendix may be used as a guide by those who wish to study the program. The overall flow of control and explanation of the more difficult parts of the program supplement the assembly listing.

A1.2 Minimum Hardware Requirements

The minimum hardware necessary for the operation of the current version of DOGGIE is:

a) PDP-8 with 8K memory

b) On-line ASR-33 Teletype

c) DEC-338 Programmed Buffered Display

d) 32K DF32 Minidisk

e) 637 Dataphone Interface to Full Duplex 201B Dataphone.

Note that no character generator is being used at the present time. A section of this appendix describes what to do to make use of a character generator if DOGGIE is run on a machine which has one.

The 637 Dataphone Interface is used to communicate with the large central computer. However, it is also used as a source of an interrupt every 3 1/3 ms. This constant rate interrupt is used to limit the refresh rate of the display and for constant-rate light pen tracking.

A1.3 Storage Requirements

DOGGIE is written for a DEC-338 with either 8K, 12K, or 16K of 12-bit core memory. The current version is assembled for an 8K machine, and the program occupies the following locations (octal):

00000 - 00004

00010 - 00015

00020 - 05777

07600 - 10002

10013 - 11343

Locations 6000 - 7577 are reserved for user programs which call upon DOGGIE. User programs may also operate anywhere in the unused areas of fields 1, 2, and 3 which have not been allocated for the storage of graph data and display file. In an 8K machine, it is recommended that user programs be limited to field 0.

The 1331_8 words of field 1 consist of unchanging display subroutines for vertex shapes, loops, arrows, and alphanumeric characters. All of these subroutines must be in the same memory field, but may be placed anywhere by suitably adjusting the origin pseudo-operations in

the assembly decks of GPACK and CHTBL. One could, for example, place these subroutines at the end of field 3.

Unused areas of fields 1, 2, and 3 may be used for the storage of graph data and display file. For any practical problem size to date, locations 11344 - 17577 have been sufficient for this purpose.

Locations 5, 6, and 7 of fields 0 and 1 are left unused so that either XOD or XDDT debugging programs may be used to aid in debugging when DOGGIE is being developed.

Locations 00016, 00017, 10010, 10011, 10012 are auto-index register locations available for user programs.

A1.4 Load, Start, Restart

DOGGIE is saved as two System Save files on the minidisk of the DEC-338 by the following commands to the Disk Monitor System:

.SAVE GPAC!00000-06177;05400

.SAVE GSYS!10000-11377;11377

In order to load and start the program, first the field 1 portion must be loaded (GSYS). When GSYS is loaded, it starts and causes the automatic loading and starting of the field 0 segment (GPAC) at 05400.

The initial routine at 05400 moves the page of code at 06000-06177 to the area 07600-07777, thus replacing the Monitor Head of the Disk System with a special one used with DOGGIE.

A switch is then set so that a later restart at 05400 will not move the Monitor Head page again.

The start and restart routine then clears software indicators of hardware flags, starts the transmitter of the Dataphone, executes the DOGGIE command RESET, and then calls upon DOGGIE to load (from disk) and start the Graph Monitor user program (GMON).

A1.5 Storage Allocation

The available areas of fields 1, 2, and 3 are divided into blocks of 19 words each. Allocation is performed by the RESET command, and a list of free blocks (available space) is generated. This is a unidirectional list with a pointer to its head.

There is a subroutine to allocate storage by successive returning of blocks to the free list. At present this storage set-up routine is called only from the RESET routine, but it could be incorporated into a more complex allocation algorithm which did not initially allocate all available storage.

There is a subroutine to get a block from the head of the free list. This routine stops at a fatal halt if it is called when there are no more free blocks.

Another subroutine returns a block to the free list by a pushdown operation. Thus the same blocks are re-used again and again even though there are other free blocks in memory. Storage is allocated such that numerically higher addresses are at the head of the free list initially.

A pointer to a block must be 14 bits in length. This is typically stored as two consecutive words: bits 7 and 8 of the first word are used as a field part and the entire second word is a 12-bit address pointer. The other bits of the first word are often used for some other purpose.

A pointer to a block does not point to the numerically lowest address of the block as is the custom in linked-list systems. Instead, a block is pointed-to at its third-last word. Since the PDP-8 has no index register, the requirements for compact storage plus speed led to this design. Figure A1-1 shows the representation of a free block. The negative and positive integers indicate relative word positions in

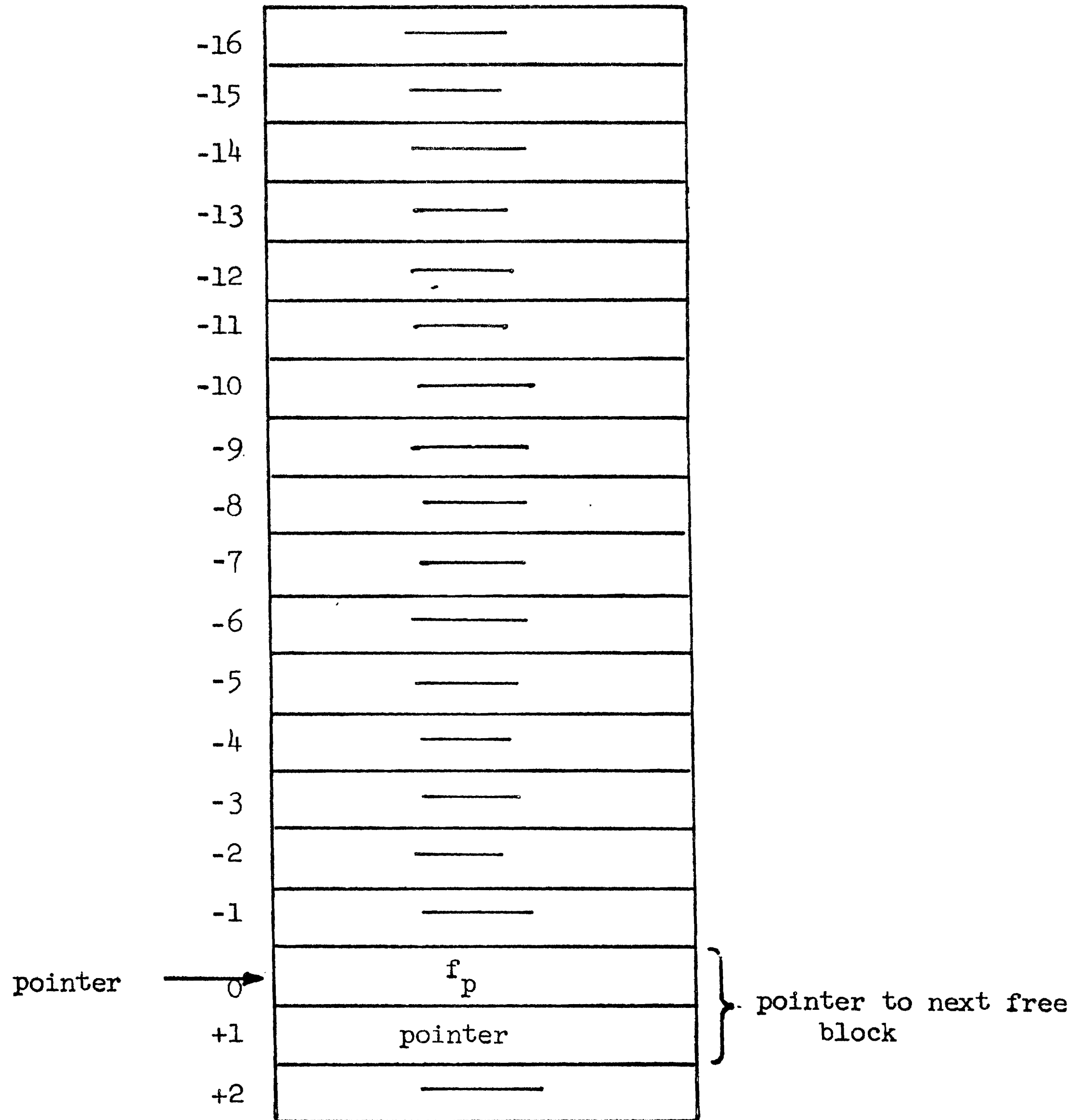


Figure A1-1 A Free Block

the block based upon the pointer location.

Note that each free block links to the next free block by a pointer in words 0 and +1. The quantity "f_p" represents the field of the pointer at bits 7 and 8 of word 0. The last free block on the free list has a pointer of all zeros as a special indicator.

The contents of the other 17 words of a free block remain unchanged from the previous use of the block.

At any particular time a block is used by DOGGIE in one of five ways:

a) free block

b) vertex block

c) arc block

d) label block

e) display-list block

A1.6 Undefined DOGGIE Commands

Commands are given to DOGGIE as sequences of 12-bit words. DOGGIE decodes these commands and dispatches to particular routines to carry out specified functions. In general DOGGIE ignores meaningless and undefined commands. The single exception of this is the halt which occurs when DOGGIE is given a command word with bit 0 as a ONE. If this happens, the user may hit CONTINUE on the PDP-8, which causes DOGGIE to suspend interpretation of its current input stream and return control to the program which called DOGGIE.

DOGGIE will properly skip over data words of a command according to the expected form of data when the vertex or arc named does not exist. For example, if a command is given to alter the paper position of a non-existent vertex, DOGGIE will skip over two data words which specify the

vertex coordinates.

A1.7 Display List

Commands to DOGGIE cause the creation, alteration, and deletion of vertices and arcs as defined within the memory of the DEC-338. One block is used for each existing vertex or arc. In addition, an extra block is required for each existing vertex label or arc label. All information specific to a particular vertex or arc is compactly stored in its one or two (when it has a label) blocks. This includes the display file code used to display the particular entity. This effective use of space is accomplished by using much of the display file code itself for the encoding of the information describing the entities and their labels.

As explained in the main text of this report, an entity may exist within the DEC-338 structure independent of its display status. The display file code of each vertex or arc is contained within its block as a display subroutine. In order for an entity to be displayed, it must be called by a PJMP display subroutine call command. A display list consisting of a list of display list blocks is maintained for this purpose. Each display list block may contain up to eight of these PJMP calls followed by a JUMP to the next display list block, except the last block of the display list may contain up to nine PJMP calls. The display list itself is a display subroutine and thus it terminates by a POP. A particular entity may be on the display list at most once. The display list is dynamic in that the display of particular entities may be started and stopped. Stopping the display of an entity causes a two-word gap in the display list where the PJMP associated with that entity's display was removed. These gaps are not closed up, but are reused when

a later entity display is begun. Due to restrictions on available memory, code to compress (or garbage collect) the display list has not been written. Therefore, the space used by the display list is determined by the maximum number of entities which have been simultaneously displayed since the last execution of the RESET command.

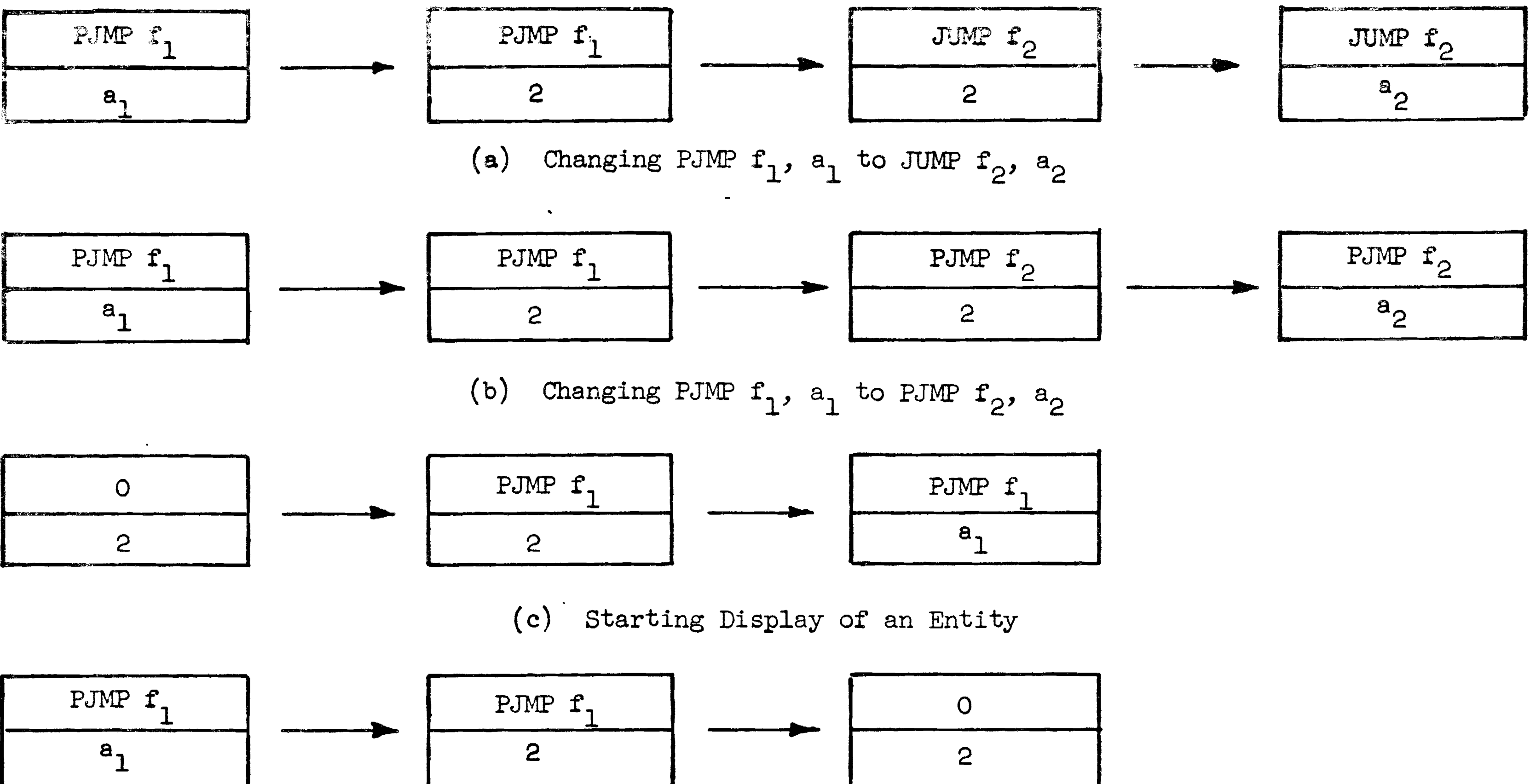
An objective of the implementation of this system has been to always keep the display running. This restriction makes alteration of the display file an operation which must be carefully planned. The display list is one such place where alteration occurs. As a sample of the techniques used to alter the display, the explanation of the maintenance of the display list is given here.

When the display list is packed to capacity (as shown in Fig. A1-2a) and another entity is to be displayed, a free block is first obtained from the head of the free list. The first pair of words is filled with a copy of the PJMP call at the end of the current display list. The next pair of words is filled with a PJMP call to the new entity to be displayed. The next seven pairs of words are filled with 0000 and 0002. Finally, the last word of this new block is set to a POP command. Figure A1-2b shows a display list block in this initial configuration.

The PJMP call at the end of the current display list is then overlaid to be a JUMP to the display list block just created. Figure A1-3a depicts the sequence of the values of this pair of words which allows for a safe transition. The underlying idea which makes such a transition work properly is location 0002 of each memory field has a POP display command. This little trick well defines each of the four forms in Figure A1-2a. A similar transition is used to alter one PJMP call to another PJMP call, as shown in Fig. A3-2b. This method is of use only if

(a) Full (terminal)	(b) Initial (terminal)	(c) Partial (non-terminal)
PJMP f_1	PJMP f_1	PJMP f_1
a_1	a_1	a_1
PJMP f_2	PJMP f_2	0
a_2	a_2	2
PJMP f_3	0	0
a_3	2	2
PJMP f_4	0	PJMP f_4
a_4	2	a_4
PJMP f_5	0	0
a_5	2	2
PJMP f_6	0	PJMP f_6
a_6	2	a_6
PJMP f_7	0	PJMP f_7
a_7	2	a_7
PJMP f_8	0	PJMP f_8
a_8	2	a_8
PJMP f_9	0	JUMP f_{10}
a_9	2	a_{10}
POP	POP	(POP)

Figure A1-2 Sample Display List Blocks



(a) Changing PJMP f_1, a_1 to JUMP f_2, a_2

(b) Changing PJMP f_1, a_1 to PJMP f_2, a_2

(c) Starting Display of an Entity

(d) Stopping Display of an Entity

Figure A1-3 Safe Display File Alteration

missing a subroutine call is allowed. This is the case in the alteration of the PJMP to a particular loop shape, arrow shape, or vertex shape subroutine.

As more entities are placed onto the display list, PJMP calls fill the terminal block until it becomes full as shown in Fig. A1-2a. Figure A1-3c shows the transitions used to fill the display list block. Then the next request for a new entity to be displayed causes another free block to be obtained, etc., as described above.

When an entity is to be removed from the display list, its associated PJMP call is removed from the display list by the transition shown in Fig. A1-3d. That word pair is then available for further PJMP calls. Figure A1-2c shows a typical non-terminal display list block with three available word pairs. "Non-terminal" means this block is not the last block of the display list. (Note it ends with a JUMP to the next display list block.)

The display list described above is a subroutine which is called from the display file driver. This driver is a sequence of display command words occupying 12 locations on page 0. It is responsible for limiting the refresh rate of the display to 23 1/3 milliseconds. This is done by waiting for pushbutton 0 to be a ONE. The PDP-8 sets this pushbutton every 23 1/3 milliseconds, and the display file driver clears it on every display cycle. The display file driver also sets pushbutton 6 to a ONE for proper display of loops. The driver includes the scale and intensity setting parameter command used for the entire graph.

A1.8 Vertex and Arc Blocks

Each vertex or arc utilizes one block for both display file and all other associated data except for a label. A vertex label or arc label occupies one block which is a subroutine called from the vertex or arc block. Vertex and arc blocks have some common properties which are introduced next. The following two sections describe the specific differences between vertex blocks and arc blocks.

All vertices existing within the DEC-338 structure are linked together by a unidirectional list of the same format as the free list. All existing arcs are linked together in the same way. This form is shown in Figure A1-4.

The field bits of the pointer are found in bits 7 and 8 of word 0 of the block. This word also includes a PNLS display command (Pop, iNhibit restoring Light pen and Scale), which does not interfere with any setting of bits 7 and 8. The last vertex (arc) in the vertex (arc) list is indicated by a pointer of 00000.

The 12-bit internal name of each vertex or arc is saved at the +2 word of the block. This is the name used in DOGGIE commands which refer to specific vertices and arcs. The vertex or arc list is scanned for such commands by a fast search routine which chases down the list looking for a matching of the name.

Although the general forms of vertex and arc blocks are very similar, only three other words of the block are similar enough to warrant mention here. The reader should refer to Figure A1-4 to see these common words.

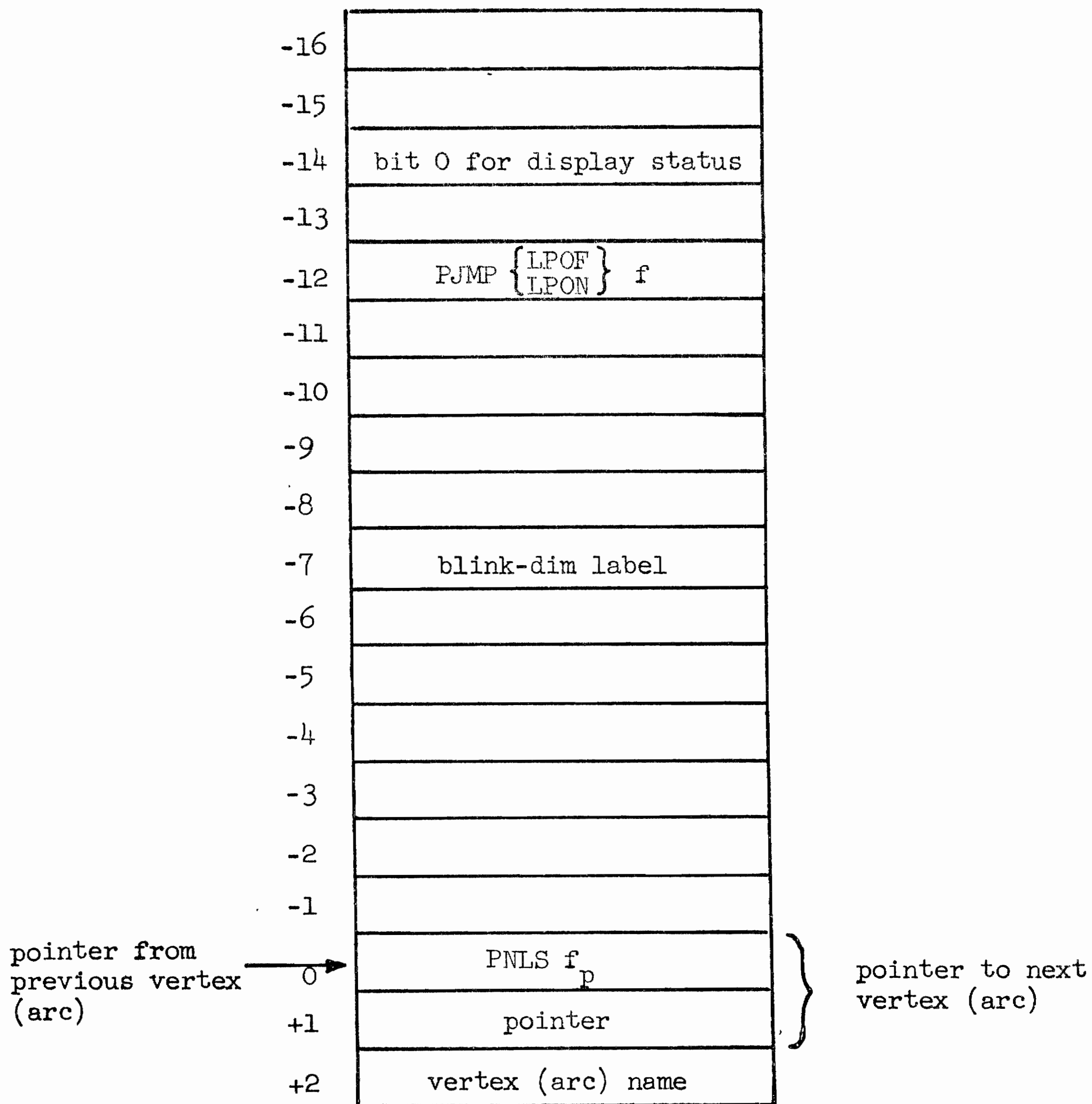


Figure A1-4 Common Properties of Vertex Blocks and Arc Blocks

Word -14 of an arc block is used for many purposes; however, one function which it has in common with a vertex block is that bit 0 of the word indicates whether that entity is being displayed by virtue of its being on the display list. This word will be again discussed for each type of entity.

Word -12 is always a PJMP with the field bits (9-11) set to the field where the block is stored because -11 is an address within the very block. Word -12 includes the unique indicator of whether the entity is light pen sensitive. This indicator is also the display file code which turns light pen sensitivity on or off.

At any one time DOGGIE is either in BLINK mode or DIM mode according to the last DOGGIE command interpreted which set the mode. The subparts of each entity may be blinked or dimmed independently of the others, and there are individual words of each vertex and arc block dedicated to this control. Word -7 of the block is always used to control the blink-dim status of the label of the entity. This word is either BKOF or BKON when in BLINK mode. It is some intensity-setting display command when DIM mode is in effect. Each vertex block contains an additional blink-dim word for the vertex shape, and each arc block includes control for blink-dim of both the arc itself and its arrow.

A1.8.1 Vertex Blocks

Figure A1-5 shows the organization of a vertex block. Four words of the 19-word block are unused in a vertex block: these are words -16, -15, -13, and -4.

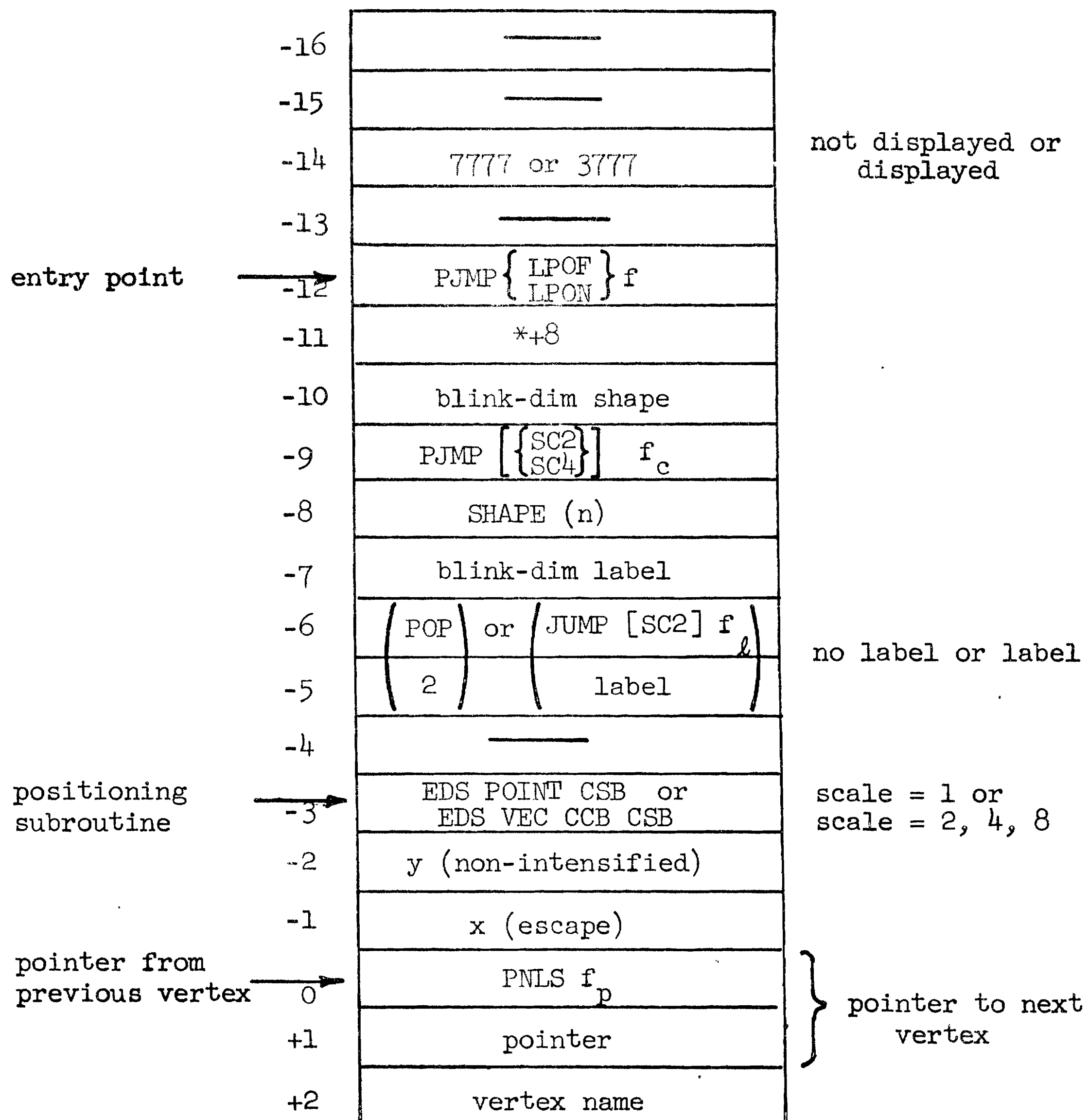


Figure A1-5 A Vertex Block

Word -14 of a vertex block is used only to indicate whether the vertex is being displayed. Bit 0 is ZERO if and only if the vertex is displayed, and the remaining bits of this word are all ONE. When a vertex is displayed there is a PJMP call on the display list to word -12 of the vertex block. The display file of a vertex ends with either a POP at word -6 or a POP within the label block associated with the vertex.

The first display command of a vertex is a PJMP call to the positioning command words of the vertex block beginning at word -3. This positioning subroutine is also called by each arc which emanates from this vertex. The PJMP at word -12 also includes the bits to control the light pen status of the vertex. If a FULL window is being used (i.e., if the display scale is 1) word -3 is EDS POINT CSB, and the next pair of words indicates the paper (or window) position of the vertex. If the window is not FULL then a vertex may be positioned anywhere on the paper, possibly off the screen area: this is done by using a non-intensified vector drawn from the origin by using EDS VEC CCB CSB followed by an appropriate Δy and Δx. These Δ's are computed according to the paper position of the vertex, the window size, and the window position. Words -2 and -1 contain the only information about the vertex position. In order for this method to work properly, the dimensions of the display are always set to correspond to the scale. For example, when the window size is FOURTH the scale is 4 and the x- and y-dimensions are set to 12 bits.

Words -10, -9, and -8 of a vertex block are dedicated to the vertex shape. Word -10 is the blink-dim word for the shape part of the vertex. The next pair of words constitute a PJMP call to the vertex shape. The eight shapes are closed subroutines ending with PNLS in the same field as character shapes, etc. The entry points for the shapes are

every other word in 16 consecutive words. Shapes are usually drawn in the same scale as the rest of the graph except when the scale is 8; in this case, shapes are drawn at scale 4 for better appearance by setting the scale as part of the PJMP. The other instance when the scale is set as part of the PJMP is when a vertex is created whose internal name is greater than 7747_8; in this case, a scale 2 setting is used. This is intended to have messages for the user in scale 2 - both light buttons and labels. Since each shape subroutine inhibits restoring the scale, and because of the way in which labels are called, any vertex whose internal name is greater than 7747_8 will be created with shape and label in scale 2. Subsequent execution of the SETWIN command will treat all existing vertices in the same way, thus eliminating the special scaling.

Words -7, -6, and -5 of a vertex block are dedicated to the vertex label. Word -7 is the blink-dim word for the label. If no offset or text have been defined then there is no vertex label block. In this case words -6 and -5 are POP and 2 respectively. If, on the other hand, there is a vertex label block words -6 and -5 constitute a JUMP to that block. The JUMP includes a scale setting for scale 2 whenever a FULL window is not being used. A description of the vertex label block is given in Sect. A1.9.

A1.8.2 Arc Blocks

Figure A1-6 shows the organization of an arc block. All words of a 19-word block are used in an arc block. It was, in fact, the arc block which dictated the choice of the block length used.

Since an arc may exist independently of the existence of its associated vertices, two words of the arc block must contain the internal names of these vertices. The names may be the same, in which case the

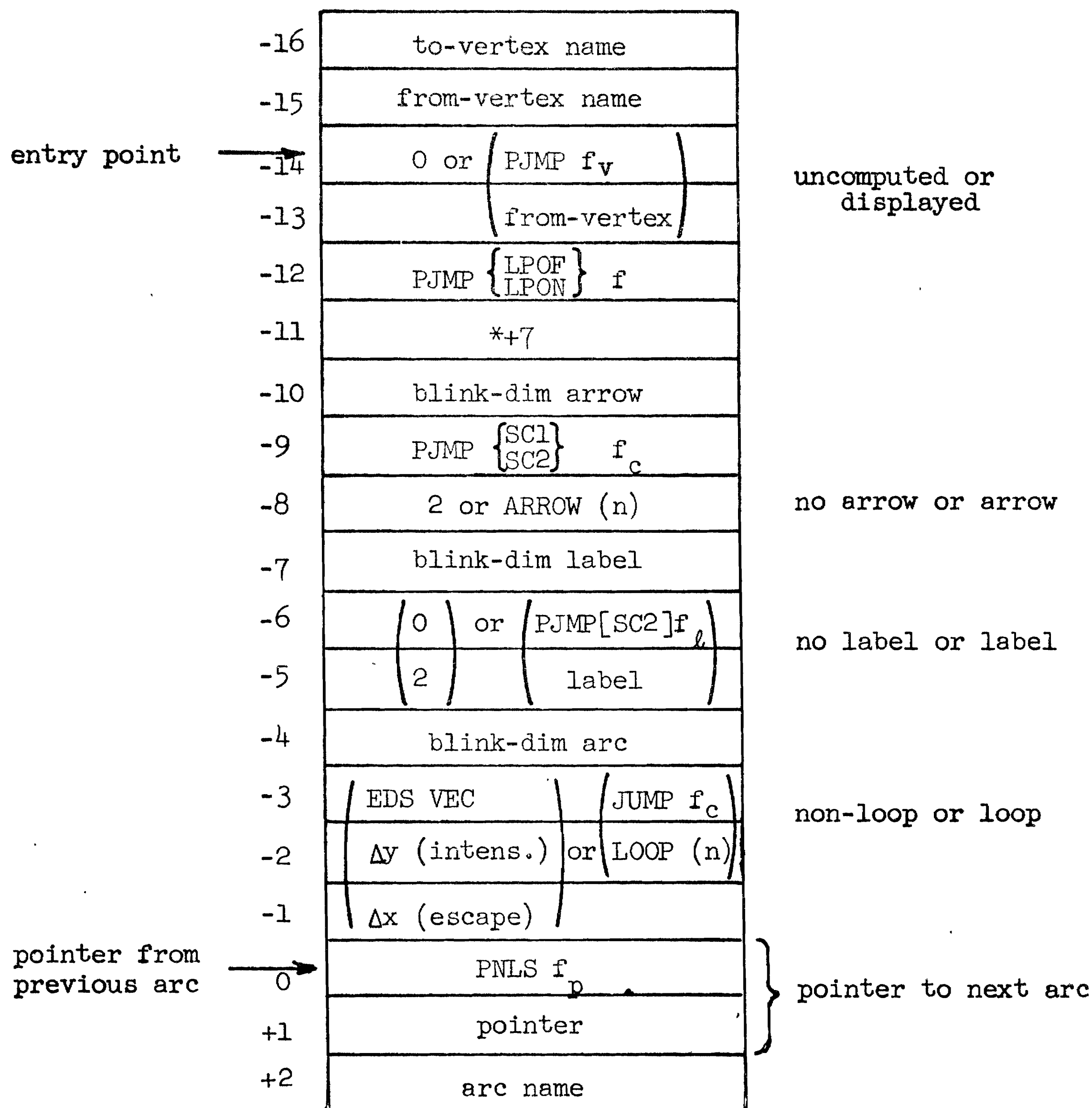


Figure A1-6 An Arc Block

arc is a loop. Word -16 contains the name of the to-vertex and word -15 contains the name of the from-vertex. The vertices have no indicators of those arcs where they are used. Instead the arc list must be scanned to find those arcs which are defined in terms of a particular vertex.

When an arc is first created, it is not automatically displayed. In order to be displayed both vertices associated with the arc must exist.

When an arc which is not a loop is displayed, it is drawn as a straight line from its from-vertex to its to-vertex. An arc which is a loop is displayed as one of the four loops available (East, North, West, South). In either case the display file of the arc must begin by a positioning of the beam to the from-vertex. When an arc is displayed there is a PJMP call on the display list to word -14 of the arc block. Words -14 and -13 then constitute a PJMP call to the positioning part of the from-vertex block (word -3).

The actual arc is then drawn in two halves. If the arc is a non-loop each half is the same vector. If the arc is a loop each half is drawn as a result of the same call on one of the loop display subroutines. Each loop subroutine, however, tests and complements pushbutton 6 upon entry and therefore can draw first and second halves on alternate calls.

The first half of an arc is drawn as a result of the PJMP call at words -12 and -11. This PJMP is made to word -4 of the arc block itself. The PJMP also includes the bits to control the light pen status of the arc. Note that word -4 controls the blink-dim of the arc itself, and this display command word is executed for each half of the arc.

After the first half of the arc is drawn, the arrow (if any) and label (if any) are drawn. Words -10, -9, and -8 of each arc block are dedicated to the arc's arrow. Word -10 is the blink-dim word for the

arrow. If an arc being displayed has an arrow, words -9 and -8 constitute a PJMP call upon one of the eight available arrows. If a displayed arc does not have an arrow these words contain a PJMP call to a POP. The PJMP includes a scale setting so that arrows are drawn in scale 1 when the scale of the graph is either 1 or 2. Otherwise they are drawn in scale 2.

Words -7, -6, and -5 of an arc block are dedicated to the arc label. Word -7 is the blink-dim word for the label. If no offset or text have been defined then there is no arc label block. In this case words -6 and -5 are POP and 2 respectively. If, on the other hand, there is an arc label block words -6 and -5 constitute a PJMP call to that block. The PJMP includes a scale setting for scale 2 whenever a FULL window is not being used. A description of the arc label block is given in Sect. A1.9.

Both the arrow and arc label subroutines preserve the beam position so that the second half of the arc can then be drawn to complete the display of the arc.

When an arc is first created it is not automatically displayed. In order for an arc to be displayed both of its associated vertices must exist. Before an arc is displayed, some of contents of the arc block are not determined. The arc is then in an uncomputed state, which is indicated by word -14 containing 0. When the display of a non-loop is begun the appropriate PJMP's must be determined for the initial positioning and for the arrow. Also the proper vector size must be determined according to the positions of the vertices. Once an arc has been computed, if it is then removed from the display list, it remains in a computed non-displayed state which is indicated by setting bit 0 of word -14 to ONE, and preserving the remainder of the word. If subsequently

there is a change which would require a re-computing of the arc, it is instead flagged as being uncomputed once again. This would happen, for example, when one of the arc's associated vertices is moved to a different position in the case when the arc is not a loop.

When an arc is a loop, its orientation is uniquely determined by which loop subroutine is called at words -3 and -2. A JUMP is all that is needed to call the loop subroutine since each one ends in a PNLS on the first half and POP on the second half.

A1.9 Label Blocks

Each vertex or arc which has an associated label offset or text has a label block. The block includes no pointers. It is entirely a display file subroutine which begins at the first word of the block and occupies as many words as are required, ending in a POP. A label block is associated with only one vertex or arc, and its existence is dependent upon the existence of its host.

There is one difference between a vertex label block and an arc label block. Although it is not necessary for a vertex label, an arc label must preserve the beam position. Therefore, an arc block must terminate with a correction vector, occupying three words of the block. These three words cannot be used for character codes as they are in a vertex label block. This is why a vertex label may be up to 27 characters in length whereas an arc label may only have 21 characters at most. Figure A1-7 shows full vertex label and arc label blocks. The subscripted a's represent 6-bit trimmed ASCII codes.

A label block begins with three words specifying the non-intensified offset vector. If a label exists, but is not displayed, the first word of the label block is a POP. The fourth word is INCR STOP, which is a

EDS VEC	
Δy (non-intensified)	
Δx (escape)	
INCR STOP	
a_1	a_2
a_3	a_4
a_5	a_6
a_7	a_8
a_9	a_{10}
a_{11}	a_{12}
a_{13}	a_{14}
a_{15}	a_{16}
a_{17}	a_{18}
a_{19}	a_{20}
a_{21}	a_{22}
a_{23}	a_{24}
a_{25}	a_{26}
a_{27}	00
POP	

Vertex Label Block

EDS VEC	
Δy (non-intensified)	
Δx (escape)	
INCR STOP	
a_1	a_2
a_3	a_4
a_5	a_6
a_7	a_8
a_9	a_{10}
a_{11}	a_{12}
a_{13}	a_{14}
a_{15}	a_{16}
a_{17}	a_{18}
a_{19}	a_{20}
a_{21}	00
EDS VEC	
Δy (non-intensified)	
Δx (escape)	
POP	

Arc Label Block

Figure A1-7 Full Label Blocks

special software signal to the PDP-8 to go into the interrupt-time routine which simulates a 6-bit character generator. This program organization will make it very easy to reassemble GPACK and CHTBL to take advantage of the speed of a character generator. The use of a character generator has one other advantage: it will make it possible to get light pen hits on labels. This is not possible now since the character generator simulator keeps initializing the display for each character, and display initialization clears the light pen enable flag.

Each character shape is stored as an increment mode subroutine. Normally a section of display file is as follows:

```
EDS INCR
  :            }   increments
POP
```

However, since the PDP-8 is performing the dispatching, each such subroutine must end in an internal stop. Instead of just replacing the POP by STOP, each POP has been eliminated and each character begins with EDS INCR STOP. An extra STOP is placed after the last character shape. Thus, the display file of characters is of the following form:

```
EDS INCR STOP
  :               }   increments
EDS INCR STOP
  :               }   increments
EDS INCR STOP

etc.
```

Thus there has been a net savings of 62 locations over the obvious organization. The only software overhead, which takes negligible time, is the necessity of performing a resume (RES2) in the PDP-8 after initializing to a new character.

The simulated character generator routine escapes whenever it reads a character code of all ZERO's in either the left or right byte. Figure A1-7 shows label blocks which contain the maximum number of text characters. When there are fewer characters (even none) the one or four termination words appear immediately following the word containing the escape code.

A1.10 Light Pen Pointing

Whenever the light pen is pointed at a displayed entity whose light pen status has been enabled, an interrupt causes the light pen handler to be entered. If the light pen handler interlock is clear, the handler will record information pertinent to the hit entity. Due to the restricted form of the display file of DOGGIE, it is very easy to determine which entity caused the light pen hit. The display pushdown list provides the fundamental key. This list consists of three levels (6 locations) where the first level simply records the call upon the display list. The second level records each call upon a displayed entity, and the third records the various subroutine calls described in Sect. A1.8. These include vertex positioning, vertex shape, arrow, arc label, and loop subroutines. The light pen handler makes use of the second level of the pushdown list to determine the entity which caused the hit.

A1.11 Light Pen Tracking

It is relatively easy to do light pen tracking on the DEC-338 using the Light Pen Sense Indicator. A square tracking box is drawn of a size barely larger than the field of view of the light pen. Before each side is drawn the LPSI is cleared. After each side is drawn, the LPSI is checked. If it is ON, this indicates that side is within the field of view of the light pen, and therefore a small correction vector is drawn which moves the box toward the side just drawn. Since this is done for

each of the four sides, it tends to keep the box positioned under the light pen as the pen is moved around the screen. A complete programming example of this method of light pen tracking is given in the report on the PDPMAP Assembly System.

The above method used for tracking is effective only if the square is drawn every few milliseconds. The less frequently the square is drawn, the slower is the speed at which the pen may be moved about the screen. This requirement makes <u>effective</u> tracking rather difficult to implement on the DEC-338. The method employed within DOGGIE takes advantage of the limited structure of the display file. If no tracking is in effect, there is no overhead of extra subroutine calls. When tracking is in progress, the PDP-8 externally stops the display every 3 1/3 milliseconds. The interrupt service routine for the external stop flag saves the contents of the second level of the display pushdown list and replaces it by a fake return to the tracking subroutine; then the display is resumed. After the display stops displaying the current entity, the tracking subroutine is entered.

The tracking routine begins by an absolute point plot for the positioning of the tracking box. After the box is drawn, with possible correction vectors, an internal stop occurs. The interrupt service routine for this internal stop reads the coordinate registers which indicate the updated tracking box position. At this time the second level of the display pushdown list is restored, and the pushdown pointer is repositioned so that the next entity to be displayed is displayed when control exits from the tracking subroutine. Thus, in effect, the tracking subroutine is like another entity on the display list squeezed between two consecutive display list subroutine calls.

The above description is meant to be an overview of the tracking process. There are some subtle special cases which arise due mainly to timing considerations for which additional checking is made.

A1.12 Use of the Disk

Since the memory size of the DEC-338 is 8K, and since much local activity is desirable, the DF32 Disk is an integral part of the Interactive Graph Theory System. This disk is used primarily for the storage of program overlays. These overlays are stored on the disk as USER SAVE files of the PDP-8 Disk Monitor System. They are loaded by a DOGGIE command according to a name of one to four alphanumeric characters. The programmer may also use the Disk System basic input/output routine to read or write 128-word blocks.

The PDP-8 Disk Monitor System is being used because of the ease of maintaining files. The author could have written his own version of the system, but such an effort would not significantly enhance the system performance. Furthermore, there are many already existing utility and systems programs which aid the Interactive Graph Theory System builder in maintaining and adding to the system. The one major disadvantage of using the already-existing Disk System is its relatively slow speed. A special-purpose rigid system might speed up current operations at least tenfold; however, this one disadvantage has been overshadowed by the many advantages of using the standard system.

All of the Disk Monitor System is used as provided by DEC except the basic input/output routine which is incompatible for use with the program interrupt system of the PDP-8. For servicing the display and and dataphone, it is an absolute necessity to keep interrupts enabled nearly all the time. The basic routine "SYSIO" has been rewritten to

operate with interrupt ON, and is compatible with the various parts of the Disk System. It can also be called by a user function program through a pointer in a communication cell on page 0. When this call is made, the real SYSIO is not called, but an alternate one is actually called which sets some indicators and then calls upon the basic routine.

The new SYSIO, along with the way in which the LOAD command is coded, allows for useful computation to proceed during disk input/output transfer times. Moreover, the interrupt service routine for the disk "remembers" one level of interrupt status, and allows other interrupts to occur during execution of the Disk System routines.

APPENDIX 2

USER PROGRAMS

This appendix supplements Chapter 4 by describing essential details regarding the creation of user programs - both local ones and those used as subroutines of the central computer. A listing of the SELECT routine described in Section 4.7.6.2.2.1 is included as an example of a user program.

A2.1 General Considerations

The PDPMAP Assembly System [27] is used to assemble user programs. The process operates in two steps: the MAP Assembler first performs the assembly through the macro and operation code definition facilities, and then a postprocessor program transforms the 36-bit code produced by the MAP assembly into 12-bit words. The postprocessor also creates off-page links to help the programmer with the paged addressing of the PDP-8. The final result of the assembly is an absolute program stored in the output file (on the disk of the IBM 7040) associated with the DEC-338 terminal. Once this binary file has been placed there, the user at the terminal may cause that file to be transmitted to the DEC-338 and have it stored on the minidisk.

Each user program source deck begins by 113 cards which serve to prime the MAP Assembler to assemble PDP-8 programs with references to the DOGGIE communication cells and the following macros: DOGGIE, ENDDOG, DOG, T. The use of these macros was explained in Section 4.6.3. For completeness, a listing of the 113 definition cards follows below. Note that the communication cell names are defined to be external symbols. The values of these symbols are defined at load time before the post-

```
        UNLIST  PDP-8 DEFINITIONS  6/1/67  (30 CARDS)               00000010
      A MACRO H,J,K,L,M,N,P   THESE 30 CARDS DEFINE THE BASIC PDP-8 00000020
      H J                     INSTRUCTION SET PLUS THE FOLLOWING ... 00000030
      K L          BY           MEMORY FIELD SETTING - 'FIELD'      00000040
      M N          M.S.         NEW OPR OR IOT DEF'N. - 'DEFINE'    00000050
        IRP P      WOLFBERG   . OCTAL ADDRESS DEF'N. - 'OCTADR'     00000060
        P                       ORIGINS -'ABSORG','RELORG','OCTORG','PAGE'00000070
        ENDM A                  REGION FOR OFF-PAGE LINKS - 'LINKS' 00000080
  LOC.0 A END.(OPSYN END)B(MACRO X)()(IRP X)((DEFINE X)ENDM(ENTRY LOC.0))00000090
        A END(MACRO X)X(BSS 0)()(USE //)((PZE -1,,X)(PMC ON)END.,ENDM)  00000100
        A LINKS(MACRO X)X(BSS 0)()(USE //)((MZE X)(USE PREVIOUS)ENDM)   00000110
        A PAGE(MACRO X,Y)Y(EQU *-LOC.0-FLD.+127)()(IFT X)               00000120
        ETC ((IFF 1'X)(DUP 2,0)(ABSORG Y/128*128)IFF(ABSORG 128*X)ENDM) 00000130
        A OCTORG(MACRO X,Y)Y(BOOL X)()(ABSORG Y)(ENDM(BEGIN //,*+16384)) 00000140
        A DEFINE(MACRO X,Y)X(OPD 10000'Y'000000,*,1,2,)()ENDM()         00000150
        A ABSORG(MACRO X,Y,Z)Y(BSS 0)Z(ORG X+LOC.0+FLD.)                00000160
        ETC ((USE //)(PZE Z,,Y)(USE PREVIOUS)ENDM(B ((AND,0)(TAD,1))))  00000170
        A RELORG(MACRO X)()(ABSORG X-LOC.0-FLD.)()ENDM((B ((ISZ,2))))   00000180
        A ()(B ((DCA,3)(JMS,4)(JMP,5)))FIELD(MACRO X)FLD.(SET X*4096)ENDM00000190
        A FLD.(SET 0)DEFINE(MACRO X,Y)X(OPD Y,,,1,)((IFF Y/7000))       00000200
      X SET Y-Y/10*2-Y/100*16-Y/1000*640-Y/7400*256                     00000210
        ENDM DEFINE                                                     00000220
        B ((OPR,7000)(NOP,7000)(CLA,7200)(CLL,7100)(CMA,7040)(CML,7020)) 00000230
        B ((RAR,7010)(RAL,7004)(RTR,7012)(RTL,7006)(IAC,7001)(SMA,7500)) 00000240
        B ((SZA,7440)(SPA,7510)(SNA,7450)(SNL,7420)(SZL,7430)(OSR,7404)) 00000250
        B ((HLT,7402)(CIA,7041)(LAS,7604)(STL,7120)(GLK,7204)(SKP,7410)) 00000260
        B ((IOT,6000)(ION,6001)(IOF,6002)(KSF,6031)(KCC,6032)(KRS,6034)) 00000270
        B ((KRB,6036)(TSF,6041)(TCF,6042)(TPC,6044)(TLS,6046)(CDF,6201)) 00000280
        B ((CIF,6202)(RDF,6214)(RIF,6224)(RMF,6244)(RIB,6234))          00000290
        A OCTADR(MACRO X,Y,Z)Z(BOOL Y)X(EQU LOC.0+Z+FLD.)(ENDM,LIST) END 00000300
```

```
        UNLIST          USER PROGRAM DEF'NS.                                00000310
        EXTERN DOGGIE,PBCLR,PBSET,PBSKIP,MANINT                             00000320
        EXTERN INTENS,WINSIZ,YWIND,XWIND,YPSEUD,XPSEUD,CHPSEU               00000330
        EXTERN KSFKSF,KRBKRB,TSFTSF,TLSTLS                                  00000340
        EXTERN BLNDIM,VERCRN,ARCCRN,HTIMER,LTIMER                           00000350
        EXTERN LPHIT1,LPHIT2,LPHIT3,LPHIT4                                  00000360
        EXTERN STAT1,STAT2,STAT3,STAT4,STAT5,STAT6,FRBLKS                   00000370
        EXTERN INTRET,SNDCH,RCVCH,DREADY,SYSIO,RESTRT,GETDAT                00000380
        EXTERN RESGET                                                       00000390
USE.    MACRO X,Y,Z                                                         00000400
Z       RELORG Y                                                            00000410
        ORG Z                                                               00000420
        USE X                                                               00000430
Y       BSS 0                                                               00000440
        ENDM USE.                                                           00000450
OCTSET  MACRO X,Y                                                           00000460
X       SET Y-Y/10*2-Y/100*16-Y/1000*128                                    00000470
        ENDM OCTSET                                                         00000480
G       MACRO X                                                             00000490
        IRP X                                                               00000500
        OCTSET X                                                            00000510
        ENDM G                                                              00000520
        G ((ALL,0000)(ALLHIT,0260)(ARC,C000)(ARROW,C040))                   00000530
        G ((ARROWD,0001)(BLINK,2400)(BLINKM,0160))                          00000540
        G ((CLRPB,0300)(COORDS,C020)(CRNAME,0000))                          00000550
        G ((CURSOR,0240)(DEFALT,0002)(DIM,2400)(DIMM,1160))                 00000560
        G ((DSPLAY,2200)(EIGHTH,0003)(EXIST,2000)(FOURTH,0002))             00000570
        G ((FULL,0000)(GOTO,1340)(HALF,0001)(LABEL,0060))                   00000580
        G ((LIST,0100)(LOAD,0350)(LOADGO,1350)(LOOP,0001))                  00000590
        G ((LOOPE,0000)(LOOPN,0002)(LOOPS,0006)(LOOPW,0004))                00000600
        G ((LTPEN,2600)(MINUS,2000)(MOVWIN,0100)(NAMES,0020))               00000610
        G ((NEXT,0000)(OFFSET,0004)(PENPNT,4000)(POSWIN,0060))              00000620
        G ((PSEUDO,0120)(RESET,0360)(SCREEN,2000)(SETCRN,0200))             00000630
        G ((SETCUR,0140)(SETINT,0020)(SETPB,1300)(SETWIN,0040))             00000640
        G ((SHAPE,0040)(START,1000)(STATUS,0220)(STOP,0000))                00000650
        G ((TEXT,0001)(TYPE,0320)(VERTEX,0010)(WHOLE,0000))                 00000660
```

```
     T MACRO Y,X          TO DEFINE SYMBOLS                              CCCCC670
    ... SET Y                                                            CCCCC680
        IRP X                                                            CCCCC690
    X'. SET ...                                                          CCCCC700
    ... SET ...+1                                                        CCCCC710
        ENDM T                                                           CCCCC720
        T 01(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z)        CCCCC730
        T 27(LB,BS,RB,UA,LA)                                             CCCCC740
        T 32(.,EX,CU,NM,DL,PC,AN,AP,LP,RP,AS,PL,CM,MI,DT,SL)             CCCCC750
        T 48(0,1,2,3,4,5,6,7,8,9,CL,SC,LT,EQ,GT,QM)                      CCCCC760
     T MACRO X            TO ASSEMBLE TEXT                               CCCCC770
    ... SET 0                                                            CCCCC780
        IRP X                                                            CCCCC790
        IFT ...                                                          CCCCC800
   .... SET 64*X'.                                                       CCCCC810
        IFF ...                                                          CCCCC820
        PZE ....+X'.                                                     CCCCC830
    ... SET 1-...                                                        CCCCC840
        IRP                                                              CCCCC850
        IFT ...                                                          CCCCC860
   .... SET 0                                                            CCCCC870
        PZE ....                                                         CCCCC880
        ENDM T                                                           CCCCC890
```

```
         DOG MACRO X,Y,Z                                          00000900
         ... SET 0                                                00000910
             IRP X                                                00000920
         ... SET ...+X                                            00000930
             IRP                                                  00000940
             PZE ...                                              00000950
             ENDM DOG                                             00000960
      DOGGIE MACRO X                                              00000970
             JMS* DOGGIE                                          00000980
             PZE X-*                                              00000990
             USE. GRAPHS                                          00001000
           X EQU *                                                00001010
             ENDM DOGGIE                                          00001020
      ENDDOG MACRO                                                00001030
             PZE              ENDS LIST                           00001040
             USE. ()                                              00001050
             ENDM ENDDOG                                          00001060
       END.. OPSYN END                                            00001070
         END MACRO                                                00001080
             BEGIN GRAPHS,*                                       00001090
             USE. GRAPHS                                          00001100
             END..                                                00001110
             ENDM END                                             00001120
             LIST                                                 00001130
```

processor operates. The definitions are contained in a separate binary deck (named GRUSER) as absolute entry points. This deck was assembled in the absolute mode (ABSMOD) without any assembled code. If the assignment of communication cells to locations ever had to change in DOGGIE, then only the GRUSER deck would have to be reassembled and postprocessing would be performed for every user program. The alternative of including the definitions of the communication cells within the 113-card definitions would necessitate a more severe changeover requiring the re-assembly of every user program.

As introduced in Section 4.6, user programs may be executed within the one area of the DEC-338 memory from location 06000 to location 07577. A user program may call upon the DOGGIE interpreter to load a program segment which co-exists in the user program area. Loading may alternatively be used to overlay a program segment in the case where the calling program is no longer needed. The programmer must beware that the loading of any program segment will not preserve the previous contents of locations 07200 through 07577.

The program interrupt system of the DEC-338 must generally remain ON during the execution of user programs. This is necessary for proper operation of the display and the various input/output devices serviced by DOGGIE. If a user program must turn interrupt OFF for some special purpose, it should be off for no longer than about one millisecond.

A2.2 Local User Programs

All of the functions available under the Graph Monitor are carried out by local user programs. The Graph Monitor itself calls upon one of nine possible program segments according to user action. Each of these segments has an initial loading address and starting address at location

06000.

Additional local user programs may be created and placed on the DEC-338 minidisk. Such programs are listed by the "Miscellaneous Functions" monitor available through the Graph Monitor. Such programs must be saved as a contiguous sequence of pages, but there is no restriction on the placement other than starting at a page boundary. When a user selects one of these functions, the "Miscellaneous Functions" monitor loads the segment and starts it at its initial load point as indicated in the directory of the Disk Monitor System. Since a local user program may include one or more program segments, the convention was established for the "Miscellaneous Functions" monitor to avoid listing any user program name which begins with a period. Thus a name of this form should be used for secondary program segments whose names should not be listed.

A2.3 User Programs as Subroutines

Section 4.7.5.2 described the facility for an interactive ALLA program in the central computer to call upon a user program in the DEC-338 as a subroutine. This section presents some information which the programmer of this type of user program must know.

There are nine available slots for user programs in the DEC-338. An interactive ALLA program may call upon one of these by the following type of statement:

```
CALL USER(n)
```

where n is an integer expression whose value is between 1 and 9. This statement causes the loading of the corresponding program segment in the DEC-338. These segments are named USE1, USE2, ..., USE9 and must

have an initial load point and starting address of location 06200. During the operation of the user program, locations 06000 through 06177 should not be altered. This area contains both the Teletype input service routine and past status information sent to the IBM 7040. In addition, it contains a termination routine which the user program may call to terminate interactive execution.

One user program, USE9, has already been written and is part of the basic system. It is used to select entities through light pen and pushbutton control. This program is presented as an example in Section A2.4. The remaining eight user program slots are available. The basic system currently contains a one-page version of each of the programs USE1, USE2, ..., USE8 so that something proper will happen in case an interactive ALLA program makes such a call. Each of these programs displays an informative message indicating that it was called. The user may then depress a pushbutton to resume interactive execution.

Within a user program, if there is to be Teletype input collected as input to the IBM 7040 via the communication cells, the programmer must be sure to call upon the Teletype service routine at location 06047 at least every 100 milliseconds.

When a user program is ready to terminate and send status to the IBM 7040, it does so by calling the DOGGIE interpreter with the following command sequence to load and start the Send-Status (SNDS) overlay:

```
DOG LOADGO
OCT 2316,0423,6200
```

A user program should include tests for the manual interrupt button. For termination of execution, the DOGGIE interpreter should be called with the following command sequence to load the Dataphone-send

routine (GSEN) and start the termination routine:

DOG LOADGO

OCT 0723,0516,6046

Examples of the above two types of termination are included in the SELECT routine listed in the next section.

A2.4 SELECT Routine

The SELECT routine is saved on the DEC-338 minidisk as user file "USE9." The characteristics of the operation of the program were described in Section 4.7.6.2.2.1. An assembly listing of SELECT produced during the first step of the PDPMAP Assembly process is given below. With the description mentioned above and the comments in the listing, there is no need for further explanation.

```
                                                UNLIST  PDP-8 DEFINITIONS  6/1/67  (30 CARDS)
                                                UNLIST            USER PROGRAM DEF'NS.

                                         *      'SELECT' INTERACTIVE USER PROGRAM ('USE9')
                                         *      USED DURING EXECUTION TO SELECT AN ENTITY
                                         *
                                                OCTADR EXIT1,6046        ENTRY POINTS IN 'SLAVE'
                                                OCTADR SERVIC,6047
                                         *
                                                OCTORG 6200
     06200   1000 04 0 06047 10001 SELECT JMS SERVIC
                                                DOGGIE
                                                DOG (START,EXIST,WHOLE,VERTEX,LIST)

 BINARY CARD 00001
     06320   000000007777     10000             OCT 7777,2024,2024
     06321   000000002024     10000
     06322   000000002024     10000
     06323   000000007776     10000             OCT 7776,2054,2024
     06324   000000002054     10000
     06325   000000002024     10000
     06326   000000000000     10000             OCT 0           END OF LIST
                                                DOG (START,EXIST,LABEL,VERTEX,TEXT,LIST)
     06330   000000007777     10000             OCT 7777,0
     06331   000000000000     10000
     06332   000000007776     10000             OCT 7776
                                                T (O,R,.,P,B,.,1,0,.,F,O,R,.,N,O,.,S,E,L,E,C,T,I,O,N)

 BINARY CARD 00002
     06350   000000000000     10000             OCT 0           END OF LIST
                                                DOG (START,DSPLAY,WHOLE,VERTEX,LIST)
     06352   000000007777     10000             OCT 7777,7776,0
     06353   000000007776     10000
     06354   000000000000     10000

                                                DOG BLINKM
                                                ENDDOG
```

```
06203  1000 03 0 06304 10001         DCA SELNEW     INITIALLY NO SELECTION
06204  1000 64 0 01400 10011 SELEC1  JMS* PBCLR     CLEAR PB 10,11
06205  0000000C7774    10000         OCT 7774
06206  1000 03 0 13400 10011 SELEC2  DCA LPHIT1     ALLOW HITS
06207  1000 34 0 06047 10001 SELEC3  JMS SERVIC
06210  1000 64 0 03000 10011         JMS* MANINT
06211  1000 05 0 06306 10001         JMP EXIT       MAN. INT.
06212  1000 01 0 13400 10011         TAD LPHIT1
06213  0000 00 0 07640 10000         SZA CLA
06214  1000 05 0 06253 10001         JMP SELLP      LP HIT
06215  1000 64 0 02400 10011         JMS* PBSKIP
06216  000000000001    10000         OCT 0001
06217  1000 05 0 06233 10001         JMP SELEC4
06220  1000 01 0 06303 10001         TAD SELNET
06221  1000 01 0 06312 10001         TAD MBLINK     REMOVE 'BLINK', -BLINK
06222  1000 03 0 13400 10011         DCA LPHIT1
06223  1000 01 0 06310 10001         TAD SAVLP3

BINARY CARD 00004
06224  1000 03 0 14400 10011         DCA LPHIT3     RESTORE
06225  1000 01 0 06311 10001         TAD SAVLP4
06226  1000 03 0 15000 10011         DCA LPHIT4     RESTORE
06227  1000 01 0 06304 10001         TAD SELNEW
06230  0000 00 0 07440 10000         SZA
06231  1000 05 0 06250 10001         JMP SELEC5
06232  1000 05 0 06204 10001         JMP SELEC1     NO SELECTION YET, IGNORE
```

```
06233  1000 64 0 02400 10011 SELEC4 JMS* PBSKIP
06234  000000000002    10000        OCT 0002
06235  1000 05 0 06207 10001        JMP SELEC3      WAIT FOR SOMETHING
06236  0000 00 0 07001 10000        IAC             PB 10 ON
06237  1000 03 0 13400 10011        DCA LPHIT1
06240  1000 01 0 06304 10001        TAD SELNEW
06241  0000 00 0 07650 10000        SNA CLA
06242  1000 05 0 06250 10001        JMP SELEC5      NO SELECTION YET
06243  1000 01 0 06303 10001        TAD SELNET
06244  1000 01 0 06313 10001        TAD MSTART      -START
06245  1000 03 0 06303 10001        DCA SELNET
06246  1000 64 0 01000 10011        JMS* DOGGIE     STOP BLINKING

 BINARY CARD 00005
06247  0 00000 0 00034 10000        PZE SELNET-*
06250  1000 03 0 14000 10011 SELEC5 DCA LPHIT2
                                    DOGGIE
                                    DOG (START,EXIST,LABEL,VERTEX,TEXT,LIST)
06360  000000007774    10000        OCT 7774,0
06361  000000000000    10000
06362  000000007775    10000        OCT 7775,0
06363  000000000000    10000
06364  000000007776    10000        OCT 7776,0

 BINARY CARD 00006
06365  000000000000    10000
06366  000000007777    10000        OCT 7777,0
06367  000000000000    10000
06370  000000000000    10000        OCT 0           END OF LIST
                                    DOG LOADGO
06372  000000002316    10000        OCT 2316,0423   SNDS
06373  000000000423    10000
06374  000000006200    10000        OCT 6200
                                    ENDDOG
```

```
06253   1000 01 0 06304 10001   SELLP TAD SELNEW        SERVICE LP HIT

  BINARY CARD 00007
06254   0000 00 0 07440 10000         SZA
06255   1000 05 0 06261 10001         JMP SELLP1
                                      DOGGIE            FIRST TIME THROUGH
                                      DOG (START,EXIST,LABEL,VERTEX,TEXT,LIST)
06377   000000007776    10000         OCT 7776
                                      T (P,B,.,1,1,.,S,E,L,E,C,T,S,.,B,L,I,N,K,I,N,G,.,O,N,E)

  BINARY CARD 00010
06416   000000007777    10000         OCT 7777
                                      T (O,R,.,P,O,I,N,T,.,T,O,.,A,N,O,T,H,E,R,.,O,N,E)

  BINARY CARD 00011
06433   000000000000    10000         OCT 0             END OF LIST
                                      ENDDOG
06260   0000 00 0 07240 10000         CLA CMA
06261   1000 03 0 06302 10001  SELLP1 DCA SELOLD
06262   1000 01 0 06303 10001         TAD SELNET
06263   1000 01 0 06313 10001         TAD MSTART        -START

  BINARY CARD 00012
06264   1000 03 0 06301 10001         DCA SELOLT
06265   1000 01 0 13400 10011         TAD LPHIT1
06266   1000 01 0 06314 10001         TAD CBLINK        BLINK
06267   1000 03 0 06303 10001         DCA SELNET
06270   1000 01 0 14000 10011         TAD LPHIT2
06271   1000 03 0 06304 10001         DCA SELNEW
06272   1000 64 0 01000 10011         JMS* DOGGIE
06273   0 00000 0 00006 10000         PZE SELOLT-*
06274   1000 01 0 14400 10011         TAD LPHIT3
06275   1000 03 0 06310 10001         DCA SAVLP3        SAVE
06276   1000 01 0 15000 10011         TAD LPHIT4
06277   1000 03 0 06311 10001         DCA SAVLP4        SAVE
06300   1000 05 0 06206 10001         JMP SELEC2        KEEP GOING
```

```
                                  SELOLT DOG (STOP,BLINK,WHOLE,ARC)      OR VERTEX
 06302  0 00000 0 00000 10000 SELOLD ***
                                  SELNET DOG (START,BLINK,WHOLE,ARC)     OR VERTEX
 06304  0 00000 0 00000 10000 SELNEW ***

BINARY CARD 00013
 06305  000000000000     10000         OCT 0              END
                                  *
                                   EXIT DOGGIE            SERVICE MAN. INT.
                                         DOG LOADGO
 06436  000000000723     10000         OCT 0723,0516      GSEN
 06437  000000000516     10000
 06440  0 00000 0 06046 10001          PZE EXIT1
                                         ENDDOG

BINARY CARD 00014
                                  *
 06310  0 00000 0 00000 10000 SAVLP3 ***                  SAVED LPHIT3
 06311  0 00000 0 00000 10000 SAVLP4 ***                  SAVED LPHIT4
                                  *
 06312 -0 00000 0 00400 10000 MBLINK MZE BLINK-EXIST
 06313 -0 00000 0 01000 10000 MSTART MZE START
 06314  0 00000 0 00400 10000 CBLINK PZE BLINK-EXIST
                                         LITORG
                                         LINKS
                                         RELORG *+2       LINKING AREA

BINARY CARD 00015
                                         END
                   00000 01111           END.

                   $DKEND SELECT
```

APPENDIX 3

ALLA PREPROCESSOR

A3.1 Introduction

The ALLA Language is compiled in two stages: the first stage consists of some minor syntax alteration, the creation of declaration statements, and some other miscellaneous functions. The input to this stage is ALLA enriched by the interactive commands; the output is a FORTRAN IV program. The second stage of the compilation process is the application of the FORTRAN compiler. The input to this stage is the output from the first stage, and the output of the second stage is a relocatable binary deck. Error detection is done in both stages. The first stage does not attempt to check for those error conditions which cause FORTRAN IV errors.

This appendix sketches the operations performed by the first stage or preprocessor. The preprocessor is a string-manipulation program. As such, it should be written in a string manipulation language capable of producing output which can be used as input to the FORTRAN compiler. SNOBOL is an effective language, but the 7040 implementation did not properly output to anything but the line printer. The L^6 Language was chosen for the writing of the preprocessor for machine independence and since it was already being used for the Data Structure Package in the Graph Theory System.

The preprocessor program consists of approximately 350 lines of L^6 source statements. The source deck is well commented and consists of over 950 card images. The program includes about 10 images which are MAP code. They are concerned with setting indicators in the L^6 Subroutine Package, referencing the DOGTBL deck, and termination. The DOGTBL deck

is a table assembled by MAP as a separate relocatable binary deck, but loaded along with the preprocessor. It is a table of all DOGGIE words along with their values.

The remainder of this appendix is a functional description of the preprocessor program. Also given is a list of error messages along with the cases which cause them to occur. The appendix ends in an example which shows a short program before and after preprocessing.

A3.2 Functional Description

The preprocessor begins by setting two switches in the L^6 Subroutine Package to cause page headings to be printed and to allow the reading of card images with a $ in the first column. All available storage is then assigned as blocks of two words, seven L^6 fields are defined, and some initialization is done which includes the printing of a heading.

The major loop of the program is then entered, which begins by reading a card image. The preprocessor operates on one card at a time. It begins by reading the card and listing it on the printed output - only the first 72 columns are read. The card is read in as a string of characters stored one character per block in a doubly-linked list of blocks. This allows for effective scanning and replacements in the card image. The card is then scanned, and in some cases transformed. If the card is not a comment card, and if no error is detected then one or more card images are outputted by using the L^6 punch operations. These "punched" cards are actually placed on the System Punch File, which during preprocessing is assigned to an area on the disk. When the FORTRAN compiler later processes the preprocessed program, it will read it directly from this disk area.

A comment card is any card with the letter "C" or the character "*" in the first column. Although these cards are listed, they are not punched as output. The first non-comment card may be a $-card. This is the only place where reading of a $-card is allowed; any other attempt causes an END card to be generated and preprocessing to be terminated.

The second non-comment card read is normally a SUBROUTINE or FUNCTION declaration card. When this card, possibly transformed, is punched, it is followed by 15 declaration cards which define the built-in functions and variables of ALLA and DOGGIE interactive statements. The cards are shown in Sect. A3.4.

In general, after a card is read, columns 1-6 are checked for any non-digits. If these columns contain a character which is neither a blank nor a digit, then an error diagnostic is printed and the card is ignored. Column 6 is used to denote a continuation card. If this column is not blank, then no scanning or transformation is done - the card is simply outputted. This means that all statements which must be transformed must be contained on one card image. This is not a severe restriction.

The card is then scanned starting at column 7. Leading blanks are ignored, and if the entire card is found to be blank, it is ignored. Beginning at the first non-blank column, up to 12 more columns are scanned. During this scanning, finding either an equal-sign, a right-parenthesis, or the end of the card causes scanning to terminate, and the card is outputted without transformation. If a left-parenthesis or blank column is encountered within these 12 columns further checking is done; otherwise the card is outputted without transformation.

If a left-parenthesis is found, any characters which may have preceded that character are checked. If they are either LELM or RELM, or the name of a property which has been previously declared as such by a PROPERTY statement, then further checking is done; otherwise the card is outputted without transformation. Further checking consists of finding an equal sign and a right-parenthesis as the first non-blank character immediately preceding it. If these tests are not met, the card is outputted without transformation. The assignment of a value to one of the elements of a pair is transformed from

LELM(expr) = ent   or   RELM(expr) = ent

to

CALL STLELM(expr,ent)   or   CALL STRELM(expr,ent)

where expr is any string of characters not including an equal-sign, and ent is any string of characters.

The assignment of a value to a property is transformed from

name(expr) = val

to

CALL SETVAL(name,expr,val)

where name is a property name, expr is any string of characters not including an equal sign, and val is any string of characters.

When a blank column is encountered within the 12 scanned columns, the preceding characters are checked. If they match one of the key words listed below further checking is done according to the particular word. Otherwise, the blank is ignored and scanning continues as if it hadn't been there. The key words which cause continued processing are:

ENTITY	DOG
PROPERTY	DOGSET
REMPROP	DOGSTRING
CREATE	DOGTEXT
INSERT	DOGFLUSH
REMOVE	GETSTATUS
PUSH	WAITCHANGE
DELETE	GETGRAPH
THROUGH	ESCAPE
END	TERMINATE

The next few subsections describe the checking and the transformations which occur for each of the statement types.

A3.2.1 Declarations

When the word "ENTITY" begins a statement, it is replaced by the word "INTEGER."

When the word "PROPERTY" begins a statement, it is replaced by the word "EXTERNAL". In addition, each property name is put onto a list of property names, which is used in the preprocessing of a statement which contains a left-parenthesis, as described in the preceding section.

A3.2.2 Data Structure Commands

The statement for the removal of a property is transformed from

```
REMPROP p FROM expr
```

to

```
CALL REMPRO(p,expr)
```

Note the dummy word "FROM" is skipped over, yet ignored. Therefore, any word may be substituted in its place. This is also the case for the INSERT, REMOVE, and PUSH statements.

The following table summarizes the transformations made on the other Data Structure commands.

FROM	TO
CREATE ATOM ent	CALL CRATOM(ent)
CREATE PAIR ent	CALL CRPAIR(ent)
CREATE SET ent	CALL CRSET(ent)
INSERT expr1 INTO expr2	CALL INSERT(expr1,expr2)
REMOVE expr1 FROM expr2	CALL REMOVE(expr1,expr2)
PUSH expr1 ONTO expr2	CALL PUSH(expr1,expr2)
DELETE expr	CALL DELETE(expr)

A3.2.3 Control Statements

The THROUGH statement is transformed in a somewhat more complex manner than the other ALLA statements described above. The general form of a THROUGH loop is:

```
        THROUGH n FORALL ent IN expr
          :
    n     :..
          :
```

This form is transformed into:

```
          CALL FORALL(d,ent,expr)
    t1    IF  (FORNXT(d)) GOTO t2
          :
    n     :..
          GOTO t1
    t2    CONTINUE
          :
```

where d is a non-zero integer indicating the depth of nested THROUGH loops, and t_1 and t_2 are generated statement numbers.

When the preprocessor encounters a bonafide THROUGH statement, it generates two unique statement numbers. These numbers start at 99001 and 99002, and two such statement numbers are generated for each THROUGH statement. The preprocessor maintains a pushdown list of pairs of statement numbers. One entry represents one open THROUGH loop, and statement numbers n and t_1 are saved.

As each card image, possibly transformed, is outputted, its statement number (if any) is checked against those statement numbers in this pushdown list. When appropriate, each THROUGH loop ending on that particular statement number is closed by outputting a pair of generated card images. The deepest loops are closed first.

This method may appear redundant, but the reader should realize that no types of GOTO statements are transformed, and it is possible, according to the description of ALLA, to transfer control out of a THROUGH loop into a higher-level THROUGH loop. The execution-time operation of THROUGH loops is discussed in Sect. A4.3.

The END statement terminates compilation (and thus preprocessing) of a program. The detection of this statement or of a $-card by the preprocessor causes a final check to be performed. The pushdown list of open THROUGH loops is observed, and if there are any open ones, an appropriate error message is printed along with a list of all those statement numbers needed to close the loops.

A3.2.4 DOG Statements

The DOG and DOGSET statements are extensively transformed in much the same way. For the DOG statement scanning begins immediately, and it results in a subroutine call statement with any number of arguments. However, scanning begins after the equal-sign for a DOGSET statement,

and it results in an arithmetic statement. In both cases, legitimate DOGGIE words are evaluated by the preprocessor, and expressions separated by commas are individually evaluated into one integer by performing the inclusive-OR of all terms of each expression. This scanning stops, and the remainder of the card image is not transformed, as soon as one of the following special characters is found: left-parenthesis, plus, minus, or asterisk.

During the scanning, if a string of characters is found which is not a DOGGIE word, it is ignored, and a corresponding error message is printed. If there is either no expression given or if the expression given in undefined, a value of 0 is assumed.

As an example, the statement:

```
DOG START EXIST WHOLE VERTEX 2,CRNAME,PENPNT SCREEN,+NEWX
```

is transformed to:

```
CALL DOG(1546,0,3072,+NEWX)
```

Also, the statement:

```
DOGSET SEWV2 = START EXIST WHOLE VERTEX 2
```

is transformed to:

```
SEWV2 = 1546
```

When the word DOGSTRING is detected by the preprocessor, the first non-blank character is found, and remembered as the signal character. If none exists, an error message is printed. The remainder of the card image is scanned, with a counter keeping track of the number of characters encountered. Scanning stops when the signal character is encountered or if the end of the card is reached. In the second case, a back scan is performed in order to find the last non-blank character on the line. Then a signal character is inserted to the right of that last non-blank

character, and the scanning and character counting is repeated.

As a result of one or two scans, a character count is then known. It is used in integer form as the first argument in the subroutine call generated. The second argument begins with the same integer count concatenated to the letter "H" and that concatenated to the character string which is between the two signal characters. Thus a statement of the form:

DOGSTRING $sc_1c_2...c_ns$

or

DOGSTRING $sc_1c_2...c_n$

is transformed to

CALL DOGSTR $(n,nHc_1c_2...c_n)$

where s is the signal character, the subscripted c's constitute the character string, and s is not the same character as any of the c's.

The other two statements belonging to this group are transformed by simply generating a subroutine call. A statement of the form:

DOGTEXT args

is transformed to

CALL DOGTEX(args)

where args represents any number of arguments separated by commas. The statement

DOGFLUSH

is transformed to

CALL DOGFLU

A3.2.5 Interactive Statements

The five statements used to control the interactive aspects of the Graph Theory System are transformed by simply generating a subroutine call.

The following table summarizes these transformations.

FROM	TO
GETSTATUS	CALL GETSTA
WAITCHANGE	CALL WAITCH
GETGRAPH g	CALL GETGRA(g)
ESCAPE	CALL ESCAPE
TERMINATE	CALL TERMIN

A3.3 Error Messages

The one type of error message which is given at the end of a source deck is the indication of those statement numbers required to end THROUGH loops. Otherwise, error messages are printed by the preprocessor immediately following the printing of the source card image which caused the error.

There are no fatal errors which stop the second phase of the FORTRAN compilation to proceed; however, the reading of a $-card after the first card of the program causes preprocessing to cease. Many errors detected by the preprocessor cause the entire card to be ignored. This is helpful in those cases where the programmer forgot to remove a mispunched card from his source deck. At the same time, this course of action may introduce other errors in the FORTRAN compilation. For example, if an ignored card had a statement number, it would most likely create undefined references to that statement number, or possibly cause a THROUGH loop to be open.

In order to be conspicuous among the listing of the source statements, error messages all begin with eight periods at the left margin. There are ten possible error messages which may appear. Along with each message is an indication of those cases which cause the message to

be printed.

(a) .... BAD LABEL, CARD IGNORED

Characters other than digits or blanks were found in the first six columns of a card which is not a comment card.

(b) .... ERROR, CARD IGNORED

Incomplete or illegal syntax has been detected for one of the following statements: REMPROP, CREATE, INSERT, REMOVE, PUSH, DELETE, DOGTEXT, GETGRAPH.

(c) .... TYPE SPECIFICATION MISSING, CARD IGNORED

A CREATE statement was missing one of the words ATOM, PAIR, or SET.

(d) .... ILLEGAL 'THROUGH' STATEMENT, CARD IGNORED

Incomplete syntax was detected or a bound variable was specified which was not a nonsubscripted entity variable.

(e) .... ILLEGAL STATEMENT NUMBER, CARD IGNORED

A THROUGH statement was found with non-digits where the statement number should be.

(f) .... ERROR IN TERM, TERM IGNORED

During the scanning of a DOG or DOGSET statement a character string was encountered which is not a DOGGIE word.

(g) .... NO '=' FOUND, CARD IGNORED

No equal-sign was found in a DOGSET statement.

(h) .... CARD IGNORED

No characters were found to be used as a string in a DOGSTRING statement.

(i) .... $-CARD ENCOUNTERED, 'END' GENERATED

A card with a dollar sign in the first column was read other than as the first non-comment card of the program, and before an END card

was read.

(j) .... THE FOLLOWING STATEMENT NUMBERS ARE NEEDED TO END 'THROUGH' LOOPS

n_1

n_2

⋮

At the end of preprocessing due to either the reading of an END card or $-card, there are THROUGH loops which have not been closed.

A3.4 An Example

A short program named MKCMPL is shown below before preprocessing (as an interactive ALLA program) and after preprocessing (as a FORTRAN IV program). Both printouts are taken from a computer run. Since preprocessing has been covered earlier in this appendix, there is no need to give further explanations. The reader may compare the two listings, turning back to the earlier sections for reference. The source program was prepared with statements beginning at column 8. The subroutine can actually be used and therefore serve as another example of an interactive ALLA program.

000610 OWOLFBOGRAPH

SOURCE STATEMENT

```
$IBFTC MKCMPL
       SUBROUTINE MKCMPL
*      MAKE THE GRAPH ON THE SCREEN A COMPLETE ONE
       ENTITY V1,V2,G
       LOGICAL SWITCH
       INTEGER INNAME
       GETGRAPH G
       DOG STOP EXIST WHOLE ARC, ALL
       DOG SETCRN ARC,1
       THROUGH 20 FORALL V1 IN LELM(G)
       SWITCH = .FALSE.
       THROUGH 20 FORALL V2 IN LELM(G)
       IF (SWITCH) GOTO 10
       IF (V1 .EQ. V2) SWITCH = .TRUE.
       GOTO 20
10     DOG START EXIST WHOLE ARC,CRNAME,+INNAME(V1),INNAME(V2)
20     CONTINUE
       DOG START DSPLAY WHOLE ARC,ALL
       CALL MESSAG(3)
       DOGSTRING 'A COMPLETE GRAPH IS SHOWN'
       TERMINATE
       STOP
       END
```

```
000610                 OWOLFBOGRAPH                        FORTRAN SOURCE LIST
        ISN            SOURCE STATEMENT

          0 $IBFTC MKCMPL  NOLIST
          1              SUBROUTINE MKCMPL
          2            DIMENSION LABEL(5),TTYBUF(11)
          3            COMMON/STATUS/UNDEF,BLNDIM,CHPSEU,WINSIZ,INTENS,YWIND,
                      1XWIND,ARCCRN,VERCRN,YPSEUD,XPSEUD,FRBLKS,LPHIT1,LPHIT2,
                      2LPHIT3,LPHIT4,STAT1,STAT2,STAT3,STAT4,STAT5,YOFF,XOFF,
                      3LABEL,TTYBUF,TTYIN,PBS
          4            EXTERNAL CRATOM,CRPAIR,CRSET,LELM,RELM,POP,
                      4PRSET,PRNAME,PRVAL,PRBCD,USESET,USETYP,USEENT,USEPR
                      5,ATOM,PAIR,SET,MEMBER,EMPTY,NULL,FORNXT,PRENT,PB
          5            INTEGER UNDEF,BLNDIM,CHPSEU,WINSIZ,INTENS,YWIND,
                      1XWIND,ARCCRN,VERCRN,YPSEUD,XPSEUD,FRBLKS,LPHIT1,LPHIT2,
                      2LPHIT3,LPHIT4,STAT1,STAT2,STAT3,STAT4,STAT5,YOFF,XOFF,
                      3LABEL,TTYBUF,PBS,CRATOM,CRPAIR,CRSET,LELM,RELM,POP,
                      4PRSET,PRNAME,PRVAL,PRBCD,USESET,USETYP,USEENT,USEPR
          6            LOGICAL TTYIN
                      5,ATOM,PAIR,SET,MEMBER,EMPTY,NULL,FORNXT,PRENT,PB
          7              INTEGER V1,V2,G
         10              LOGICAL SWITCH
         11              INTEGER INNAME
         12              CALL GETGRA(G)
         13              CALL DOG(1024, 0)
         14              CALL DOG(128,1)
         15              CALL FORALL(1,V1,LELM(G))
         16 99001 IF(FORNXT(1))GOTO 99002
         21              SWITCH = .FALSE.
         22              CALL FORALL(2,V2,LELM(G))
         23 99003 IF(FORNXT(2))GOTO 99004
         26              IF (SWITCH) GOTO 10
         31              IF (V1 .EQ. V2) SWITCH = .TRUE.
         34              GOTO 20
         35 10           CALL DOG(1536,0,+INNAME(V1),INNAME(V2))
         36 20           CONTINUE
         37           GOTO 99003
         40 99004 CONTINUE
         41           GOTO 99001
         42 99002 CONTINUE
         43              CALL DOG(1664,0)
         44              CALL MESSAG(3)
         45              CALL DOGSTR(25,25HA COMPLETE GRAPH IS SHOWN)
         46              CALL TERMIN
         47              STOP
         50              END
```

APPENDIX 4

ALLA MEMORY STRUCTURE AND SUBROUTINE PACKAGE

A4.1 Introduction

This appendix describes the implementation underlying the execution of ALLA programs. It serves as the counterpart to Chapter 3 which described ALLA at the level of the programmer. The ALLA programmer need not be acquainted with the information described here, in the same sense that the FORTRAN programmer need not know the underlying details of the FORTRAN system. This appendix is included for the reader interested in memory structures and for the systems programmer maintaining the ALLA System.

A4.2 Memory Structure

This section describes the way in which the IBM 7040 memory is allocated for the storage of the ALLA Data Structure as described in Chapter 3. The Data Structure is descriptive of the relationships being modeled as observed at the level of the programmer of ALLA. Although the data structure influences the way in which the storage is used, there is much flexibility over the design of the memory structure. The memory structure is presented here as it is currently implemented in the L^6 system. The reader of this section is expected to be acquainted with L^6 on the IBM 7040 as basically described in a paper by Kenneth Knowlton and further advanced by the author and Paul A. T. Wolfgang.[32,77]

All available storage is allocated as blocks of two 36-bit words for purposes of handling the ALLA structure. Six of the 72 available bits of each block are used by the L^6 system as indicators; these are bits 0-2 and 18-20 of word 0 of each block. Since pointers must be 15 bits in field width, this is not a burden.

A4.2.1 Rings

The fundamental constituent of the memory structure is the ring which is used to model sets, property-sets, and use-sets. Each ring has at least one block called the ring-start. The ring consists of a ring-start and any number of links (link blocks) which are all tied together by a doubly linked list. Figure A4-1 shows a ring with three link blocks. The shaded areas of the blocks indicate unused bits. The small letters within the blocks indicate L^6 field names. The B field always contains a back pointer, and the F field is used as a forward pointer so the whole ring is doubly linked. In a ring with no links, both the B and F fields of the ring-start point to the ring-start itself. The H field of each link block contains a pointer to the ring-start of the ring in which the link is contained. The E field of each link contains a pointer to the block which is the actual element of the ring; the link blocks themselves are used only to relate the elements.

The T field of most blocks used in the structure is a 3-bit indicator of the type of block. This field is 0 for link blocks, and it may be 1, 3, 4, or 5 to indicate the various categories of ring-starts.

A4.2.2 Entity Blocks

Each entity (atom, pair, or set) existing in the structure is represented by a single block - an entity block. Each entity block has a P field which is in general a pointer to the property set of the entity. An empty property set may be indicated by the P field containing 0. Each entity block also has a U field which is in general a pointer to the use-set of the entity. An empty use-set may be indicated by the U field containing 0. Figure A4-2 shows the entity blocks for each type of entity.

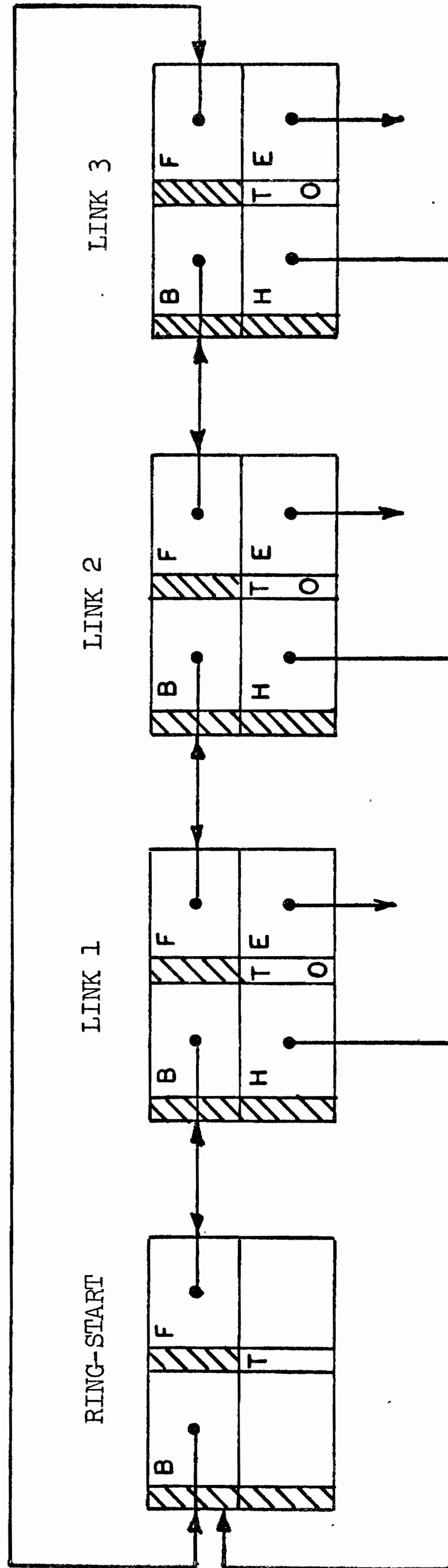
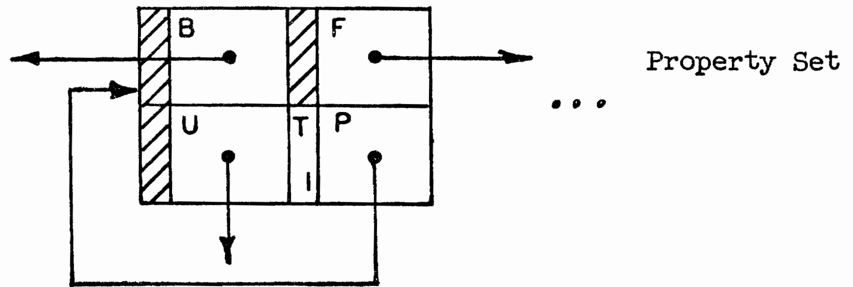
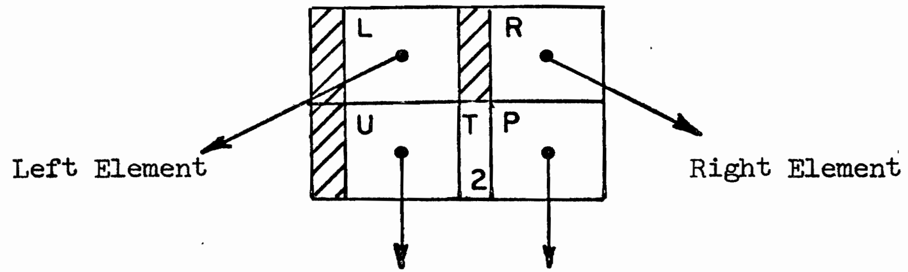


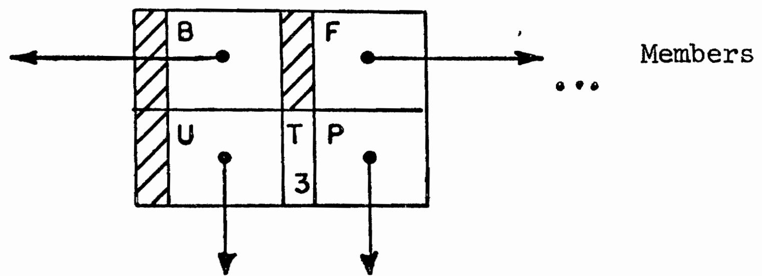
Figure A4-1 A Ring with Three Links



(a) Atom



(b) Pair



(c) Set

Figure A4-2 Entity Blocks

An atom entity block is distinguished by a T field of 1. Since an atom has no substructure, the property set of an atom is the atom itself. As shown in Fig. A4-2(a), the P field of the block points to the block itself. An atom entity block is therefore always a ring-start. When the property set of an atom is empty the B field and F field also point to the block itself.

A pair entity block is distinguished by a T field of 2. If the pair has a left-element the L field contains a pointer to it; otherwise the field contains 0. If the pair has a right-element the R field contains a pointer to it; otherwise the field contains 0.

A set entity block is distinguished by a T field of 3. This type of block is always a ring-start for the ring of members of the set. When the set is empty the B field and F field of the block point to the block itself.

A4.2.3 Property Sets

The previous section described how an atom is its own property set. The property set of a pair or a set is a separate ring pointed to by the P field of the entity block. This kind of ring is distinguished by a T field of 4 as shown in Figure A4-3. Field P of the ring-start of the property set is a pointer to the entity block of the entity to which the property set belongs. Note this is consistent with the arrangement of the property set of an atom.

The elements of a property set are called property elements. Each property element is represented by one property element block. Its form is shown in Figure A4-4. Each property element block is referenced only once; this is as a particular element of a particular property set. Field N of a property element block is a representation of the property

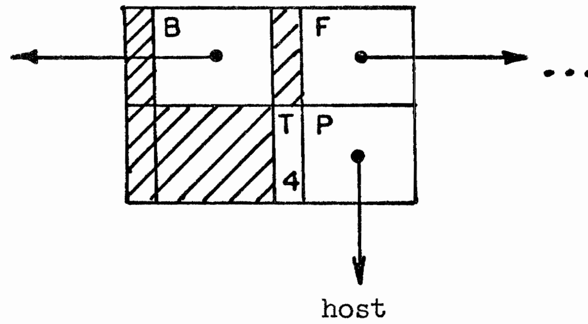


Figure A4-3 Property Set of a Pair or Set

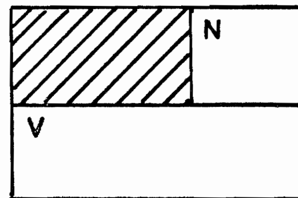


Figure A4-4 A Property Element Block

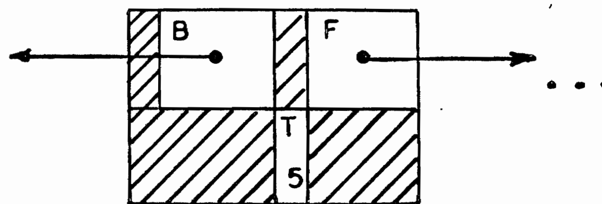


Figure A4-5 A Use-Set

name of the property element. It is the 15-bit address of the location in the IBM 7040 memory which is the entry point of the function whose name is the property name in question. It is this function which is called when this property of an entity is used. Field V of a property element block is the 36-bit value of the property of the property element. When the property is an entity property, only the low-order 15 bits are meaningful; they constitute a pointer to the entity which is the value of the property element.

A4.2.4 Use-Sets

Section A4.2.2 described how entities point to use-sets (see Figure A4-2). A use-set is represented by a ring with the T field of the ring-start containing 4. For completeness, the form of a use-set is given in Figure A4-5.

The elements of a use-set are called use-elements. Each use-element is represented only by the element pointer in field E of each link block of the use-set. As described in Section 3.17 there are three types of uses which a use-element may represent:

1. When a use-element represents a use as the value of an entity property, the corresponding element pointer points to the link block (of the property set) which points to the property element where the use occurs.

2. When a use-element represents a use as an element of a pair, the corresponding element pointer points to the pair where the use occurs.

3. When a use-element represents a use as a member of a set, the corresponding element pointer points to the link block (of the set) which accounts for the particular membership.

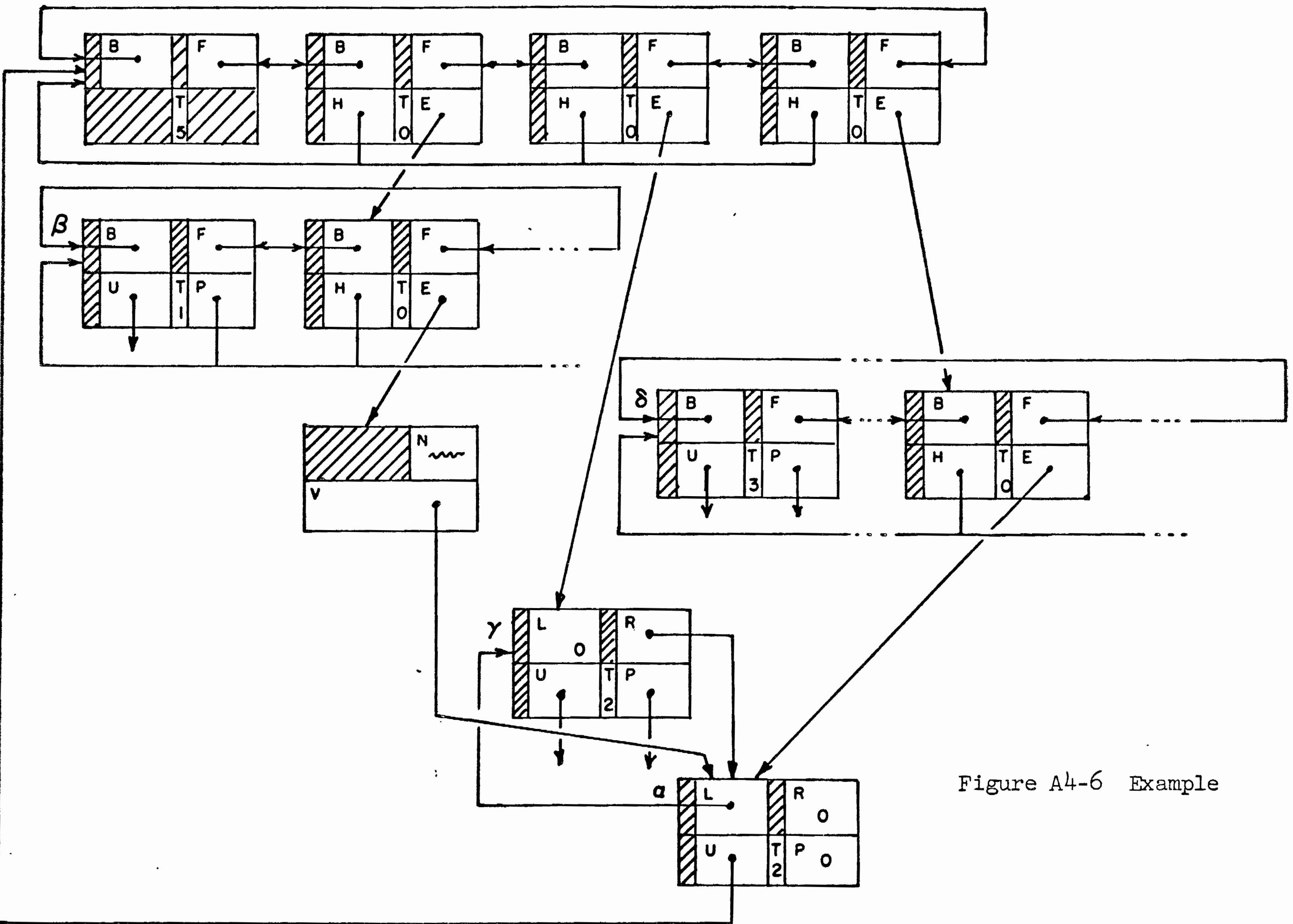


Figure A4-6 Example

To illustrate the above cases, an example is presented in Figure A4-6 where a pair α is used as a value of a property of an atom β, as a right-element of a pair γ, and as a member of a set δ. The figure shows only enough blocks to be of help in illustrating the structure of the use-set. Note γ also happens to be the left-element of α.

A4.3 Subroutine Package

The Subroutine Package is the collection of subroutines which define the ALLA data structure and memory structure. The preprocessor transforms ALLA into FORTRAN IV, so that standard FORTRAN CALL statements and LOGICAL and INTEGER FUNCTIONs are used to call upon the subroutines of the ALLA Subroutine Package.

The Subroutine Package is part of the symbolic deck DATSTR which is compiled by L^6. The deck is a mixture of MAP macro definitions, L^6 source code, and some IBM 7040 assembly language. The size of the deck is nearly 600 cards including comments, where about one-third of the deck is L^6 code. There are 30 subroutines written to be compatible with FORTRAN calling sequences. These are written in L^6 except for the subroutine definition and return statements. These subroutines call upon about 12 common subroutines written in L^6 alone. The deck also includes the initialization routine which allocates free storage, defines L^6 fields, etc. One other subroutine included in the DATSTR deck is written in a mixture of L^6 and MAP assembly code; it is used in the evaluation of properties.

A4.3.1 Form of Subroutines

Five macros are defined which allow for the interfacing of FORTRAN with L^6. A subroutine which is called as a FORTRAN SUBROUTINE begins with a statement of the form

SUBR name($a_1,a_2,\ldots,a_n$)

where name is the name of the subroutine, each subscripted a is a single letter representing an L^6 bug name, and n is the number of arguments. Argument lists in FORTRAN consist of pointers to actual arguments. The SUBR macro converts each argument pointer into an L^6 pointer (by complementing it) and stores it in the bug named by the corresponding letter in the SUBR statement. Thus the L^6 routine is called by name for each argument which is either a variable name or a FUNCTION name. A subroutine of this form ends with a conventional RETURN statement of the form

RETURN name

where name is the name of the subroutine.

A subroutine which is written to be called as a FORTRAN FUNCTION begins with a statement of the form

FUNC name($a_1,a_2,\ldots,a_n$)

which defines a FUNCTION subroutine in the same manner that the SUBR statement defines a SUBROUTINE. The one difference between these two types of subroutines is that a FUNCTION subroutine returns a result to the calling program via the accumulator register of the IBM 7040. In order to specify such a return, the following statement form is used:

ACRETN name,a

where name is the name of the FUNCTION subroutine, and a is a single letter representing the L^6 bug name of the bug whose contents are to be placed in the accumulator. This version of a FUNCTION return is appropriate for INTEGER FUNCTIONS.

A LOGICAL FUNCTION subroutine is terminated by one of the following statement forms:

```
        T.RETN name
or      F.RETN name
```

where name is the name of the FUNCTION subroutine. The first form signifies a returned value of .TRUE. and the second form is for a value of .FALSE. .

A4.3.2 Examples

The three subroutines presented in the following subsections are the actual ones being used in the Subroutine Package. The first one is the INTEGER FUNCTION subroutine which evaluates the left-element of a pair. The second one is the LOGICAL FUNCTION subroutine which determines whether its argument is an empty set. Both of these subroutines call upon the common subroutine CHECKA which determines whether bug A points to a word containing 0 and thus represents the undefined entity. If this is the case, the alternate return is used. When the normal return is taken, bug A contains the argument itself. Note field A represents the address portion of the word, i.e., it is defined as bits 21-35 of word 0. The source listing of this subroutine follows:

```
CHECKA THEN (A,A)
       IF (A,E,O) FAIL
       THEN DONE
```

The third subroutine presented is the INTEGER FUNCTION subroutine used to create an atom. This routine is included among the examples since it makes use of the call-by-name feature whereas the first two examples call upon the CHECKA subroutine, thus immediately evaluating the argument.

A4.3.2.1 LELM

The source listing of LELM follows:

```
        FUNC LELM(A)
        THEN (LELM1,DO,CHECKA)
        IF (AT,N,2)THEN(A,E,0)LELM1
        THEN (A,L)
LELM1   ACRETN LELM,A
```

The returned value of LELM is 0 (undefined) when either the argument is undefined or is not a pair; otherwise the contents of the L field of the block representing the pair is returned.

A4.3.2.2 EMPTY

The source listing of EMPTY follows:

```
        FUNC EMPTY(A)
        THEN (EMPTY1,DO,CHECKA)
        IFALL (AT,E,3)(AF,P,A)EMPTY1
        F.RETN EMPTY
EMPTY1  T.RETN EMPTY
```

The value of EMPTY is .TRUE. if and only if either its argument is undefined or is a set which is empty.

A4.3.2.3 CRATOM

The CRATOM subroutine calls upon the common subroutine CRRNG which creates a ring-start. The source listings of CRRNG and CRATOM follow:

```
CRRNG   THEN (A,GT,2)(AF,P,A)(AB,P,A)DONE

        FUNC CRATOM(X)
        THEN (DO,CRRNG)(AP,P,A)(AT,E,1)(XA,P,A)
        ACRETN CRATOM,A
```

A4.3.3 Maintaining Structure

Section A4.2 described the memory structure being used for the storage of the ALLA Data Structure. This section offers insight to the dynamics of the operations performed on the structure.

The process of creating an entity is rather straight-forward as presented in Sect. A4.3.2.3. Somewhat more involved is the building up of the structure by the routines: INSERT, PUSH, STLELM, STRELM, and SETVAL. The latter three routines may have to remove a use-element from a use-set before forming the new relationship. As the new relationship is then formed, by any of these routines, a use-element must also be created except in the cases when an element of a pair is made undefined or the SETVAL routine is used for a non-entity property.

The destruction of the structure requires the most work for maintenance as exemplified by the length of the DELETE subroutine, which is 39 source cards. When an entity is deleted, its use-set is scanned and each instance where the entity was being used in the structure must be deleted. For example, an entity being deleted must be removed from all sets of which it is a member. Of course, the use-set itself must also be deleted. The property set of an entity being deleted must also vanish. Each property-element must be checked since a corresponding use-element must be removed for each deleted entity property. Another burden which the REMOVE and DELETE routines must perform is the checking for those ways active THROUGH loops might be affected. This is further explained in Sect. A4.3.5. Each routine responsible for structure destruction returns to the L^6 storage allocator all blocks being removed from the structure. Thus, if the ALLA programmer is careful to delete unused portions of the structure, there is no garbage build-up as system

overhead.

A4.3.4 Properties

Each property name being used in an ALLA program must have an associated three-word subroutine in the deck named PROPS. This deck is a MAP deck which includes the definition of the PROP macro. This macro is called as many times as necessary to define property names, where each call may define as many names as will fit on one source card. All entity property names must be defined first. The entry point PROP.P indicates the start of the deck by tagging the first property. The entry point PROP.N indicates the break between entity properties and non-entity properties by tagging the first non-entity property. The entry point PROP.E indicates the end of the deck by tagging a word of all ZEROS immediately following the last property. The MAP code generated for each property name is of the following form:

```
        ENTRY name
name    TRA **
        TSL PROP.Ø
        BCI 1,name
```

where name is the property name. The subroutine PROP.Ø is included in the DATSTR deck, and is coded in both MAP and L^6. The first two assembled words for each property (second and third lines above) constitute a function which is used whenever the property of an entity is evaluated within any expression.

The SETVAL routine is used to set the value of a property of an entity. As described in Sect. A3.2, the ALLA statement

```
name(expr) = val
```

is transformed to

```
CALL SETVAL(name,expr,val)
```

by the preprocessor. This presupposes that the property name name was previously declared as such in a PROPERTY declaration statement. Each PROPERTY statement is transformed to an EXTERNAL statement by the preprocessor, which causes the call upon the SETVAL routine to include as the first argument a pointer to the entry point of that property within the PROPS deck.

In the handling of property elements by the ALLA programmer, it is possible to have a variable whose value is the pointer to the property or a BCD code of the property name. For this reason the routines SETVAL, REMPRO, PRENT and PRBCD first determine whether their first argument points to a location within the bounds PROP.P to PROP.E; if not, the argument is assumed to point to a location containing the pointer, or it may point to a location containing the BCD code of a property name. In the latter case a scan of the property names must be performed in order to yield a property address.

A4.3.5 THROUGH Loops

For purposes of reference, the preprocessing of a THROUGH loop is repeated here. The general form of:

```
        THROUGH n FORALL ent IN expr
          :
   n     ...
          :
```

is transformed into:

```
        CALL FORALL(d,ent,expr)
   t1   IF (FORNXT(d))GOTO t2
          :
   n     ...
        GOTO t1
   t2   CONTINUE
          :
```

where d is a non-zero integer indicating the depth of nested THROUGH loops, and t_1 and t_2 are generated statement numbers (see Sect. A3.2.3).

In order to keep track of both the bound variable and the sequencing through a set, an L^6 list is maintained at execution time. This list is unidirectional, using field F as a pointer. The list is headed by bug T, and thus, it is called the T list. The end of the T list is indicated by 0. This list has one block for each level of THROUGH loop currently in use. Bug L contains an integer indicating the current level or depth. Initially, both bugs L and T contain 0. Each block on the T list contains two data items: field H points to the bound variable being used for the associated loop, and field E points to either the ring-start or one of the links of the ring representing the set being scanned for the associated loop.

The FORALL subroutine controls the blocks of the T list according to the contents of bug L and the first argument of the call upon the FORALL routine. This is the vehicle used to determine when control is transferred to somewhere outside the range of a THROUGH loop. Each call upon the FORALL routine causes a new block on the T list to be initialized, where the E field contains a pointer to the ring-start of the set to be scanned. Then each call upon the corresponding FORNXT routine advances the E field pointer around the ring by the forward pointers of the link blocks of the ring. As long as there is another member, the value of the FORNXT routine as a LOGICAL FUNCTION is .FALSE., which causes the statements within the range of the loop to be executed. In this case the bound variable is made to point to the member of the set corresponding to the link pointed to by the E field pointer. The value of the FORNXT routine is .TRUE. when there are no more members in the set being scanned,

and thus control passes out of the range of the loop.

The T list just described must also be checked by the subroutines REMOVE and DELETE since they may affect the composition of a set being scanned by one or more THROUGH loops. The E fields of the blocks on the T list point to links of sets being scanned according to the most recently used member. If a REMOVE or DELETE statement causes the removal of one of these link blocks, any E field pointers of the T list must be backed up to the previous link (or ring-start). In the special case when a set currently being scanned by a THROUGH loop is deleted, each E field pointer of the T list which is pointing to one of links of this deleted set is set to 0. The next application of the FORNXT function will recognize the 0 as a condition to leave the range of the loop.

APPENDIX 5

INTERACTIVE EXECUTION

This appendix describes the methods employed for communication between the central computer and the graphics terminal during interactive execution. There is also a description of the basic system employed in the IBM 7040. Listings of two basic interactive ALLA subroutines are included.

A5.1 Methods of Interaction

This section is meant to clarify the operation of the various subroutines in the deck DOGDOG and the various program segments in the DEC-338 used during interactive execution. Knowledge of the INTSUP program in the IBM 7040 is assumed, as described in "The Input/Output and Control System of the Moore School Problem Solving Facility."[46]

Interactive programs are executed on the IBM 7040 at internal console number 1, which is the second console assigned to the DEC-338. Through the Graph Monitor and the program segment EXECUT, the user indicates the parameters for execution using the Teletype as an input device. It prepares a MULTILANG job for him which consists of a call upon the IXSYS worker program.[45] After appending the job to the input file, EXECUT displays the message "WAITING FOR EXECUTION," and issues the console command START.

Next, the EXECU1 program segment is loaded and started, and it requests output. The expected response at this point is to wait. An end-of-file response would be an error. When output does appear, the program checks that it is only one line which is the IXSYS calling sequence: "IXSYS/GRAPH". If there is any more or less output than

this, it is considered an error. In the event of no errors yet, the previous message is replaced by "PROGRAMS ARE BEING LOADED".

Normally, at this point, there is at least thirty seconds of loading time while the IBSYS Loader operates. If an error occurs during loading, output will be placed on the output file which will be detected as an error condition, since the EXECU1 program is waiting for an end-of-file condition. Such a condition is created when execution actually begins since the READY-BIT and RUN-BIT are turned off just before the L^6 initialization is performed.

Next, the EXECU1 program issues the console commands "C 0" and "START 2". Again, it then waits for output. If and when output appears, it must be the coded command for GETSTATUS. Otherwise, an error condition arises.

An error condition due to any of the above situations causes the Teletype bell to ring twice and the display of the blinking message "ERROR IN EXECUTION". The user is then given the choice to use the text console.

Once a good GETSTATUS code is detected, the previous message is replaced by "EXECUTION BEGUN", and the program segment SLAVE is then loaded and started. It first types a 'RETURN' and then loads and starts the program segment SNDSTA which sends status to the IBM 7040. As status is being sent, a 'LINE-FEED' is typed. After this, the program segment SLAVE1 is loaded and started which then waits for more DOGGIE commands. This is the normal operation of the DEC-338 during interactive program execution.

DOGGIE commands, including fourteen special GOTO commands for interaction, are outputted to the output file 110 IBM 7040 words at a time. This process may occur as fast or slowly as necessary. Since the output file is of finite length, there is control over a maximum number of records placed on the output file. When the DOGGIE command buffer is flushed, if that maximum number (150 is used currently) has been exceeded, the RUN-BIT and READY-BIT are both turned off and an indicator switch is set. At the next attempt to output a buffer to the output file the program waits for the READY-BIT to become set.

Meanwhile the SLAVE1 program in the DEC-338 continues to request output, and feeds DOGGIE words to the interpreter. If it ever reaches the end of the output file, this is because the RUN-BIT and READY-BIT are both off. It then issues the console commands 'C O' and 'START' which will trigger the outputting of DOGGIE command buffers again. The SLAVE1 program then returns to its normal mode of operation of requesting output.

Whenever an interactive statement is executed in the IBM 7040 except for GETGRAPH or TERMINATE, a return of status is expected. This is obtained on the input file. The subroutine in the IBM 7040 waits indefinitely for a change in the APPEND POINTER at location S.PLIN-1. When this occurs, it reads in the twenty-two words of status information, updates the communication cells, and returns to the calling program.

Since a graph may be of arbitrary size, the READY-BIT is used to indicate that the coding of the entire graph of DOGGIE has been appended to the input file. This is done by the program segment SAVGR1 which issues a console command "START" after having appended the entire graph.

In order to stop execution, the user may hit the manual interrupt button, which transfers control in the DEC-338 to the termination routine beginning in the program segment SLAVE. An interactive user program may also transfer control to there. The termination routine first issues the console command "CIDS", which causes the console's READY-BIT to be set in location S.PFIL-1 in the IBM 7040 and also the tag of that word is set to 1. The "CIDS" command is a variant of the "START" command. Next, one input line is appended which contains only 12 bits of ZERO. This is followed by the issuing of the "STOP" command. Normally, the "STOP" command would be sufficient, but in the IBM 7040, INTSUP will not cease the execution of a job when its RUN-BIT is off. Therefore, in those sections of the subroutines in the IBM 7040 where the RUN-BIT may be off, the tag of location S.PFIL-1 is checked for possible termination of execution.

A5.2 The Basic System

The previous section described the way in which an interactive execution job is prepared for processing by MULTILANG on the IBM 7040. An IXSYS macro is called which causes the loading of those binary decks which correspond to the description specified by the user. For example, if the user typed "SHORT" as a description of the program decks, then all binary decks in the data file with an associated key-word (or descriptor) "SHORT" would be included as part of the load. In addition, the macro causes all decks with the key-word "G.SYS" to be loaded; this is what defines the binary decks which constitute the basic system to be loaded each time. Although the basic system now includes six binary decks, a system programmer may add another deck to the data file and

add the key "G.SYS" to it. This deck would then become part of the basic system.

In order to not be misleading, it should be mentioned that the IBSYS Loader obtains about 15 binary decks from the Relocatable Subroutine Library to become a part of the loaded basic system.

The six decks which are tagged with "G.SYS" are:

1) DATSTR - the package of subroutines underlying the ALLA Data Structure, and the initialization routine (see Appendix 4).

2) PROPS - the defined properties which may be referenced (see Sect. A4.3.4).

3) GRAPIN - the subroutine for reading in a graph (see Sects. 4.7.7.1 and A5.3).

4) INOUT - the subroutine for computing incoming and outgoing arcs (see Sect. 3.19.2).

5) ERROR - the standard subroutine for termination with an error message (see Sects. 4.7.8.2 and A5.3).

6) DOGDOG - the package of subroutines responsible for the interactive components of the interactive ALLA language plus a few miscellaneous subroutines coded in MAP. The following subroutines are contained within the DOGDOG deck: DOG, DOGSTR, DOGTEX, DOGFLU, CLRPB, SETPB, BITS, BITSIN, ITOA, ATOI, GETSTA, WAITCH, GETGRA, SELECT, ESCAPE, TERMIN, USER, PB, GET12, GET12I (see Sect. 4.7). The STATUS control section is defined within this deck; it contains all of the communication cells defined for each interactive ALLA subroutine (see Sect. A3.4).

A5.3 Listings of ERROR and GRAPIN

This section contains the listings of two interactive ALLA programs which are part of the basic system. The ERROR subroutine was described in Sect. 4.7.8.2, and the GRAPIN subroutine was described in Sect. 4.7.7.1.

```
      SUBROUTINE ERROR(TEXT)
      INTEGER TEXT(5),I,ITOA,T(5)
      DOG 0,0,0,0, TYPE, 7, 7, -1
      CALL MESSAG(6)
      DOGSTRING 'ERROR IN EXECUTION'
      CALL MESSAG(5)
      DO 20 I=1,5
20    T(I) = ITOA(TEXT(I))
      DOGTEXT T(1),T(2),T(3),T(4),T(5)
      TERMINATE
      STOP
      END
```

```
      SUBROUTINE GRAPIN(G)
      ENTITY G,VERTS,ARCS,ENT,V
      INTEGER MISCG,TEMP,TEMP1,TEMP2(5),NAME,I,J,BITS,BITSIN,OR
      INTEGER  GMISC,GYWIND,GXWIND,CRNAMS,YCOORD,XCOORD,OFFSET
      INTEGER  INNAME,GDIM,LABEL1,LABEL2,LABEL3,LABEL4,LABEL5
      PROPERTY GMISC,GYWIND,GXWIND,CRNAMS,YCOORD,XCOORD,OFFSET
      PROPERTY INNAME,GDIM,LABEL1,LABEL2,LABEL3,LABEL4,LABEL5
      CALL GET12I
      G = UNDEF
      CALL GET12(MISCG)
      IF (MISCG .EQ. 3120) RETURN
      LELM(CRPAIR(G)) = CRSET(VERTS)
      RELM(G) = CRSET(ARCS)
      GMISC(G) = MISCG
      CALL GET12(TEMP)
      GYWIND(G) = TEMP
      CALL GET12(TEMP)
      GXWIND(G) = TEMP
      CALL GET12(TEMP)
      CALL GET12(TEMP1)
      CRNAMS(G) = BITSIN(12,23,TEMP) + TEMP1
   10 CALL GET12(MISCG)
      CALL GET12(NAME)
      IF (NAME .EQ. 0) RETURN
      CALL GET12(TEMP)
      CALL GET12(TEMP1)
      IF (BITS(32,32,MISCG) .EQ. 0) GOTO 11
      INSERT CRATOM(ENT) INTO VERTS
      YCOORD(ENT) = TEMP
      XCOORD(ENT) = TEMP1
      GOTO 20
   11 INSERT CRPAIR(ENT) INTO ARCS
      N = 0
      THROUGH 15 FORALL V IN VERTS
      IF (INNAME(V) .NE. TEMP) GOTO 13
      RELM(ENT) = V
```

```
      N = N + 1
13    IF (INNAME(V) .NE. TEMP1) GOTO 15
      LELM(ENT) = V
      N = N + 1
15    IF (N .EQ. 2) GOTO 20
      CALL ERROR(8HGRAFIN-1)
20    GMISC(ENT) = MISCG
      INNAME(ENT) = NAME
      IF (BITS(24,24,MISCG) .EQ. 1) GOTO 30
      CALL GET12(TEMP)
      GDIM(ENT) = TEMP
30    IF (BITS(29,29,MISCG) .EQ. 0) GOTO 10
      CALL GET12(TEMP)
      CALL GET12(TEMP1)
      OFFSET(ENT) = BITSIN(12,23,TEMP) + TEMP1
      DO 70 I = 1,5
      TEMP2(2) = 0
      TEMP2(3) = 0
      DO 40 J = 1,3
      CALL GET12(TEMP2(J))
40    IF (BITS(30,35,TEMP2(J)).EQ.0) GOTO 50
50    TEMP = OR(BITSIN(0,11,TEMP2(1)),BITSIN(12,24,TEMP2(2)),TEMP2(3))
      GOTO (61,62,63,64,65),I
61    LABEL1(ENT) = TEMP
      GOTO 70
62    LABEL2(ENT) = TEMP
      GOTO 70
63    LABEL3(ENT) = TEMP
      GOTO 70
64    LABEL4(ENT) = TEMP
      GOTO 70
65    LABEL5(ENT) = TEMP
70    IF (BITS(30,35,TEMP).EQ.0) GOTO 10
      CALL ERROR(8HGRAFIN-2)
      END
```

Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
University of Pennsylvania The Moore School of Electrical Engineering Philadelphia, Pa. 19104	UNCLASSIFIED 2b. GROUP NA

3. REPORT TITLE

AN INTERACTIVE GRAPH THEORY SYSTEM

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

Technical Report

5. AUTHOR(S) (First name, middle initial, last name)

Michael S. Wolfberg

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1969	350	78

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
Nonr 551(40) b. PROJECT NO. c. d.	MS 69-25 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None

10. DISTRIBUTION STATEMENT

Reproduction in whole or in part is permitted for any use of the United States Government.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research Information Systems Branch Washington, D. C.

13. ABSTRACT

The medium of computer graphics provides a capability for dealing with pictures in man-machine communication. Graph Theory is used to model relationships which are represented by pictures and is therefore an appropriate discipline for the application of an interactive computer graphics system. Previous efforts to solve Graph Theoretic problems by computer have usually involved specialized programs written in a symbolic assembly language or algebraic compiler language.

In recent years, graphics equipment with processing power has been commercially available for use as a remote terminal to a large central computer. Although these terminals typically include a small general purpose computer, the potential of using one as a programmable subsystem has received little attention.

These motivations have led to the design and implementation of an interactive graphics system for solving Graph Theoretic problems. The system operates on an IBM 7040 with a DEC-338 graphics terminal connected by voice-grade telephone line. To provide effective response times, computing power is appropriately divided between the two machines.

The remote computer graphics terminal is controlled by a special-purpose executive program. This executive includes an interpreter of a command language oriented towards the control of existence and display of graphs. Several interactive functions such as graph drawing and editing are available to a user (Cont'd.)

DD FORM 1473

Security Classification

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Binary deck						
Computer graphics						
Data structure						
Finite state machine						
Graph theory						
Graphic display						
Mealy model						
Multilang						
Multilist						
Property element						
User program						

Cont'd.

through light button and pushbotton selection. These functions which are local to the terminal are programmed in a mixture of the terminal computer's machine language and the interpreted command language.

For more significant computational requirements the central computer is used, but response time for interactive operation is then diminished. In order to overcome the speed of the telephone link, the central computer may call upon a program at the terminal as a subroutine.

Based on the mathematical terminology used to define graphs, a high level language was developed for the specification of interactive algorithms. A growing library of these algorithms provides routines to aid in the construction and recognition of various types of graphs. Other routines are used for computing certain properties of graphs. Graphs may be transformed by some routines with respect to both connectivity and layout. Any number of graphs may be saved and later restored.

A programmer using the terminal as an alphanumeric console may call upon the programming features of the system to develop new interactive algorithms and add them to the library. Programs may also be created for the display terminal, using the central computer for assembly.

Examples of system use which are presented include finding a shortest path between any pair of vertices in a weighted directed graph, determining the maximally complete subgraphs of an arbitrary graph, interpreting a graph as a Mealy model of a finite state machine, and laying out a tree for aesthetic presentation.