SLAC-134
STAN-CS-71-227
UC-32
(MISC)

GEMS - A GRAPHICAL EXPERIMENTAL META SYSTEM *

JAMES EDWARD GEORGE
STANFORD LINEAR ACCELERATOR CENTER
STANFORD UNIVERSITY
Stanford, California 94305

PREPARED FOR THE U.S. ATOMIC ENERGY
COMMISSION UNDER CONTRACT NO. AT(04-3)-515

August 1971

* Work supported in part by National Science Foundation Contract No. 2SFGJ687.

ABSTRACT

The implementation of graphical languages and graphical systems has become too complex to permit economical experimentation with many new languages or systems. Further, many applications function only as an interactive stand alone system or as a slave system; some are further restricted to particular input or output devices.

A model for graphical systems with a linguistic base is presented; the model provides symmetry between recognition and generation of pictures, although emphasizing generation. This model facilitates a more economical experimentation with graphical systems with a linguistic base and provides device independence. A graphical system defined utilizing GEMS can function interactively or as a slave system.

The model is implemented by defining its components utilizing a simple precedence translator writing system. This implemented graphical model is illustrated by two applications. First, a two dimensional mathematical expression display system is defined and implemented using GEMS. And second, a drawing system for synthesizing digital pictures for pattern recognition experiments is also defined and implemented using the model. The use of both implementations is illustrated in both the interactive and slave modes; device independence is also illustrated for both applications.

ACKNOWLEDGMENTS

I wish to thank my thesis advisor, Professor William F. Miller, for providing the intellectual encouragement which made this research possible.

Many people contribute in various ways during the course of a project; it is impossible to acknowledge all, but I would like to mention interactions with Dr. Ira Pohl, Dr. Lance J. Hoffman, Charles T. Zahn and, especially, Dr. Alan C. Shaw.

A special thanks to Professors Edward J. McCluskey and Cleve Moler and Dr. Alan Kay for their constructive readings of this thesis.

This work was supported in part by the U. S. Atomic Commission Contract AT(04-3)-515 and the National Science Foundation Contract No. 2SFGJ687.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis is concerned with the development, implementation and application of a model for graphical systems which generate pictures by computer. The model emphasizes generation, but allows a restricted class of recognition problems to be considered.

1.1 History of Project

After Shaw had developed his Picture Description Language (PDL) for pattern recognition (Shaw 1968a and b; Miller and Shaw 1967), I became interested in applying these techniques to the generation problem (i.e., picture generation).

An interactive generation version of PDL was implemented (George 1967c, 1968; George and Miller 1968) on the IBM 360 system using the IBM 2250 Display. This implementation was more difficult than first anticipated; most of the difficulty was directly attributable to processing the I/O for the 2250 Display. The problem is due to the low level detailed programming necessary to utilize this device. Further, this programming is usually not applicable to other devices, thereby resulting in an interactive application being constrained to a particular device.

In addition, the interactive PDL generation version is inconvenient for many applications; for example, those areas with a high degree of connectivity in the anticipated pictures. This inconvenience results from PDL being restricted to precisely one tail and one head as possible points to concatenate pictures. Many real elements have more than two connection points (for example, transistors have three; transformers can have four or more) and PDL is quite cumbersome to use for these types of applications.

At this point, I became interested in experimenting with languages with a linguistic base of which PDL is an example. I could not afford the investment in system development for each experiment that was required for the interactive PDL, hence I needed to develope a model of these systems which would indicate how to define a meta system, within which they could be defined. A typical interactive graphic system is illustrated in Fig. 1.1. This example can be partitioned into a control component, a data structure, a language to manipulate the data structure, a communication interface and a graphic display interface; deriving a graphic display from the data structure generally involves a low level graphics language as illustrated in the figure.

These partitions can be abstractly defined for a class of graphical systems; the partitioning is illustrated in Fig. 1.2 where each component represents the definition of a template. For the class of systems under consideration, the control component determines the actions to perform upon receipt of the proper input; the communication is accomplished through the use of an interface template. The definition of the data structure also defines a program for interactively specifying an instance of the data structure. The linear string graphic language is defined by a syntactic and semantic component.

1.2 Goals

This initial trial implementation and partitioning led to the design of GEMS (George 1969a and b). The goals for this design were:

1. Facilitate an economical experimentation (programming time necessary to implement an application) with graphical systems with a formal linguistic base;
2. Provide a symmetrical model for generation and recognition, although emphasizing generation;

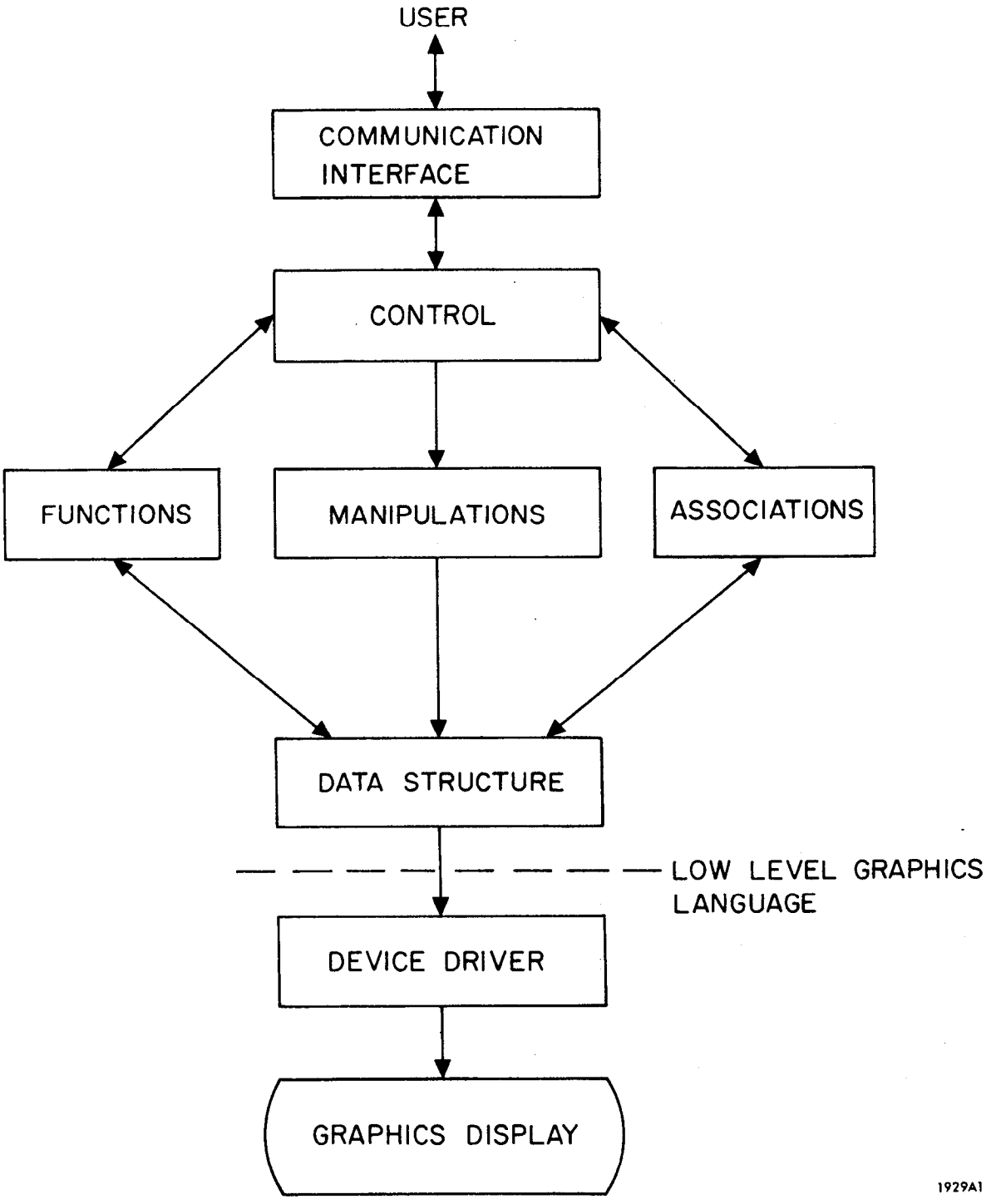


FIG. 1.1--A typical interactive graphic system.

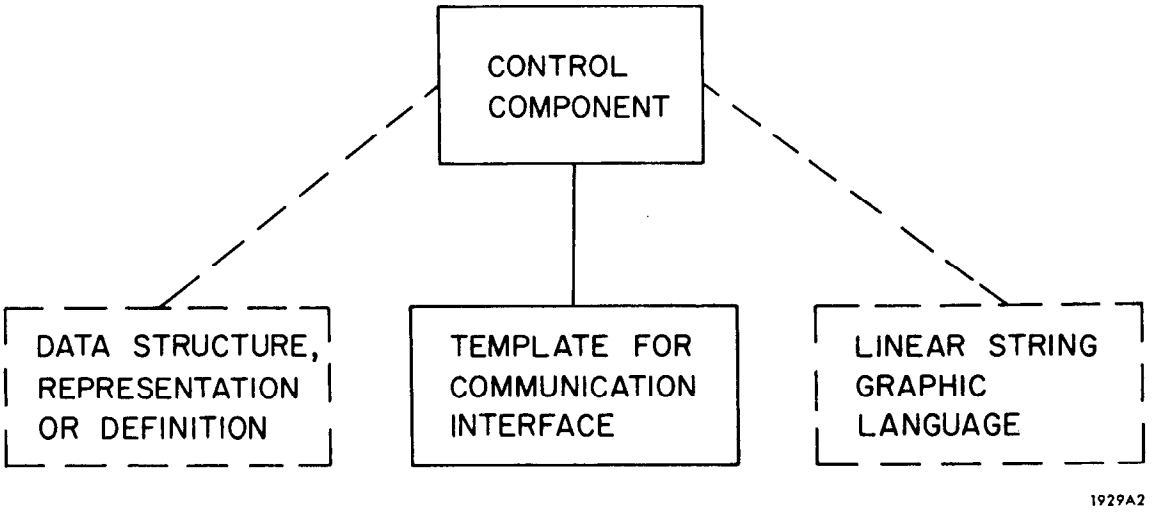


FIG. 1.2--The GEMS model.

3. Provide device independence and ease of device usage;
4. A system implemented using this model should be able to function interactively and in a slave mode;
5. Illustrate the power which can be gained by including a translator writing system within a high-level language.

1.3 Contributions

The main contribution is the development and demonstration of a model for graphical systems with a formal linguistic base which provides for economical experimentation with these languages in a non-timesharing environment; the model encourages the user to define his system so that it is device independent. Also, during the course of this research, a simple precedence translator writing system was implemented with an error recovery mechanism and techniques were developed for removal of precedence conflicts (George 1971b); the translator writing system was used to define the model presented in this report as well as other systems.

CHAPTER 2

A BRIEF SURVEY †

There have been several surveys of linguistic methods in computer graphics (Feder 1966a; Miller and Shaw 1968; Rosenfeld 1969b) and one article which has a discussion on meta systems and graphic meta systems (Miller 1969). Only those articles which bear directly on models for graphical systems will be discussed.

Picturelab-An Interactive Facility for Experimentation in Picture Processing (Arthurs, et. al. 1970)

The Picturelab system is interactive, command driven, permits symbolic variable referencing, permits the creation and execution of symbolic command files, permits storing of partially processed pictures and is usable interactively as well as a batch program. It is intended for research in processing digitized pictures utilizing visual interaction to analyze the digitized data.

Interactive Graphics and Computer Aided Design Techniques (Baskin 1970)

Baskin argues that "the cost of applications programs for a new graphic application has become the prohibitive factor in limiting the growth of the practical use of computer graphics in industry and education". He then proposes (and is currently implementing and experimenting with) a design facility to define computer aided design problems. He further divides a graphical problem into five phases; defining primitives, modeling of the problem, analysis of the problem, defining new analysis procedures and output of the results. His goal is to provide a general system with which a user can solve

† The bibliography contains as accurate a compilation as possible of those articles which I have reviewed during the course of this research.

his entire design problem independent of the particular field or special languages used in the design process.

I agree that the cost of a graphic application should be reduced, but believe that trying to solve the user's entire problem is not the answer; we should first try something more modest, like determining the desirable features of graphical languages.

A System for Implementing Interactive Applications (Chen and Dougherty 1968)

Chen and Dougherty suggest that graphics programming would be easier if:

1. Basic display images could be specified in a high-level language;
2. Interactive controls could be specified in a high-level language;
3. The resultant system was device independent.

Then, they report on their system which is intended for application to general graphical problems in computer aided design, however the reported system is not device independent. Their main point of including a graphical capability and adequate control mechanisms within a high-level language is well taken (see Dahl 1968 and Fisher 1970 for further discussions of control mechanisms within high-level languages).

SAP: A Model for the Syntactic Analysis of Pictures (Inselberg 1968)

Inselberg presents a model for both recognition and generation. In his model, pictorial data is restricted to scenes which are composed of figures which are built from primitives combined by relations. The primitives are a circle, an ellipse, a rectangle, an isosceles triangle and four differently oriented right triangles. The relations are: is on top of, is under, is to the right of, is to the left of, is contained within and contains.

The model is composed of a syntactic component and a semantic component. The syntactic component determines the structural description (i.e. a string description of the picture in terms of primitives and relations) of the input picture. The semantic component then classifies this structural description, according to the user's specified grammar, as a particular scene if it is a legal construct. A picture may also be generated directly from a structural description.

Also, he claims "the possibility of merging generation and recognition into one totally integrated system provides for further learning capabilities plus a high degree of man/machine interaction" and that "the use of the structural description (i.e. a linear string) is an efficient way to store pictorial data".

The primitives of a system should not be so restricted, further some efficiency can be obtained by a closer coupling of the classification and the syntactic component (Shaw 1968a).

A General Purpose Graphic Language (Kulsrud 1968)

Kulsrud proposes that a general purpose graphical language be developed which is device independent and can handle both generation and recognition. Further, it is suggested that a meta compiler be used to construct a compiler for the general purpose language so that it be available on various machines. The meta compiler system is illustrated for a generative type graphical language. The main problem will be in the definition of the general purpose graphical language with these characteristics and not in the specific implementation method.

Problems of Defining and Compiling Graphic Languages (Morpurgo and Sami 1970)

Morpurgo and Sami point out that the main problems relating to graphics in a batch-processing environment are: defining a language to describe the manipulations desired; choosing a data structure; and, once the language has been "accepted", implementing an "efficient" compiler for it. Further, one of the main problems in interactive graphics is the subdivision of work between a remote terminal and the main processor. They suggest that graphic languages may be defined by extending a general purpose language (for example see Hurwitz, Citron and Yeaton 1967) or creating new languages by writing a complete compiler or a meta compiler for a class of languages.

A System for Interactive Graphical Programming (Newman 1968)

Newman presents a model for interactive programs which separates each application into a procedure component and a control component. He then presents a graphical system for defining the control component utilizing state diagrams; a node represents a state and the branches represent action taken in response to data input or interrupts. A compiler for state diagrams is used to compile the control component. The system is only available interactively and is highly device oriented.

GULP - A Compiler-Compiler for Verbal and Graphic Languages (Pankhurst 1968)

In Pankhurst's system, a language is defined by production rules with imbedded semantics; the semantics are those defined in GULP or external subroutine calls. Thus, a user is not restricted in what he can do, but he must do it in an assembly language. The system is good for specifying interrupt processing but is inconvenient for specifying program computation. He does

remain essentially device independent, by generalizing the notion of a character to a linear string of characters and including interrupts within this generalization.

On the Interactive Generation and Interpretation of Artificial Pictures (Shaw 1969b)

Shaw presents a model for graphics which is quite close to the one presented in this thesis (my work had its beginnings in Shaw's early work and there have been many discussions between us since). Pictures are specified in a virtual space by symbolic descriptions; display generation occurs by executing these descriptions and mapping the resulting picture into the real space of the display device; computations over pictures are also defined.

SKETCHPAD: A Man-Machine Graphical Communication System (Sutherland, I. E. 1963)

SKETCHPAD was one of the first graphical systems which received a wide reception and generated wide interest in computer graphics. It focused a welcome and timely attention on computer graphics and probably is primarily responsible for computer graphics today. The system allowed constraints and was used for varied applications. His point that it is not worth the effort unless something more than just a picture is obtained is related to Baskin's cost of an application.

CHAPTER 3

A MODEL FOR GRAPHICAL SYSTEMS

The GEMS model is primarily aimed at interactive graphical systems with a formal linguistic base (i.e. those systems with a linear string graphical description facility such as PDL, (Shaw 1968a and b; Miller and Shaw 1967)). However, it does allow non-interactive systems to be defined and tested interactively; the defined system can then be used non-interactively without programming changes.

A typical graphical system would consist of a control description C, a definition description D (or a primitive definition description) and a graphical description G; D and G are optional but are normally expected.

The class K of graphical systems which can be described using GEMS is formally defined by

$$K = \bigcup_{\substack{C \in L_C \\ D \in L_D \\ G \in L_G}} (C, \{D\}, \{G\}) \dagger$$

Where

L_C is the language for control descriptions

L_D is the language for definition descriptions

L_G is the language for graphic descriptions

L_C, L_D and L_G will be defined for a particular implementation in Chapter 4.

† { } indicates that the item may appear zero or more times.

3.1 The Control Description

C, the control description, describes in L_C what sequence of instructions is to be performed upon the recognition of a particular input; several applications call this a menu function (Chen and Dougherty 1968) however, control description is closer to its function of recognizing menu items and, then, performing all the processing before recognizing further menu items. Many of the input messages are processed by the program generated from C; it is also the responsibility of this program (hence, of C also) to maintain the desired display. C, the control description, is defined by

$$C = (C_F, [C_C]) \dagger$$

Where

C_F is the functional part of C

C_C is the constructional part of C

There is no fundamental difference between C_F and C_C; C_F is intended for menu-like items and C_C is intended for those items related to the graphical description. The division does allow these to be functionally separate and to be displayed separately.

3.2 The Definition Description

D, the definition description, provides the facility for defining primitive graphical elements interactively within the general primitive representation. A primitive is defined by ††

$$\text{PRIMITIVE} = (\text{NAME}, [T_S], T_V)$$

† [ ] indicates an optional item.

†† The primitive representation is a slightly modified version of Shaw (Shaw 1968a and b, 1969b; Miller and Shaw 1967).

Where

NAME is a unique name

T_S is the terminal syntactic string (For a basic primitive, T_S would be null; if the primitive were composed from other primitives then T_S would be a string describing this composition.)

T_V is the terminal semantic string

T_V has two components; an attribute list and a BDF (Basic Display File). † The attribute list consists of name value pairs; the BDF consists of constructional items such as lines or arcs and the proper parameters.

Example

For a primitive straight line in Shaw's PDL (Shaw 1968a), from (0,1) to (2,3) with tail and head at the end points, the values could be:

NAME = A

T_S = null

ATTRIBUTE LIST = TAILX=0 TAILY=1 HEADX=2 HEADY=3

BDF = LINE 0 1 2 3

T_V = ATTRIBUTE LIST ; BDF

PRIMITIVE = NAME ; T_S ; T_V

A picture, P, will consist of two of the components of a primitive,

$$P = (T_S , T_V)$$

Thus, functionally there is no difference between a picture and a primitive.

Given a description, D, of those elements of a primitive to be defined, the translator for L_D constructs a program which requests the elements in order, checks each element for proper type, builds the primitive and saves it in an

† The BDF is in many cases a virtual display which must be translated to a particular display device; this provides the desired device independence.

area reserved for primitives; it also can function in a slave mode. In the slave mode, the input string is assumed to be the primitive and is saved with no data checking.

3.3 The Graphic Description

G, the graphic description, is defined by

$$G = (G_S \ , \ \langle G_V \rangle \ ) \quad \dagger$$

Where,

G_S is the syntax description of the desired graphic language; and

G_V is the associated semantic description for the syntax G_S.

G_S describes in L_G how to parse the terminal string T; for each parsing step, G_V specifies the semantics to be performed. The result of applying the descriptions of G to T is to determine the missing component of T (i.e. either T_S or T_V). Thus, generation is characterized in Fig. 3.1; recognition is characterized in Fig. 3.2. From the analogy to arithmetic expressions, evaluation of T_S to obtain T_V is generation and the evaluation of T_V to obtain T_S is recognition.

This model of generation and recognition is different from many conventional models, however, it results in a symmetry between the two and provides assistance in the areas which can be formalized without a significant loss of generality. This symmetry allows generation and recognition to coexist within the same application. Kulsrud and Shaw (Kulsrud 1968; Shaw 1968a and b) have proposed that this symmetry would be useful; the author also shares this opinion.

Fig. 3.3 illustrates the display of a graphic element (a triangle) on a particular display device. The graphic device driver translates T_V into the

† $\langle \ \rangle$ indicates that the item may occur one or more times.

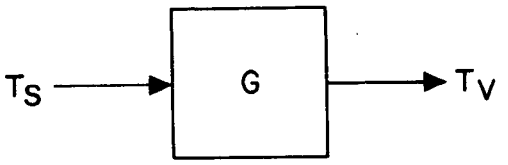


FIG. 3.1--Generation.

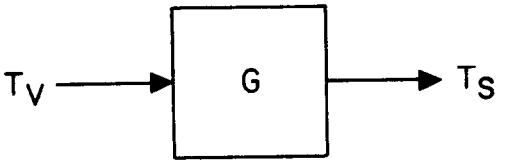


FIG. 3.2--Recognition.

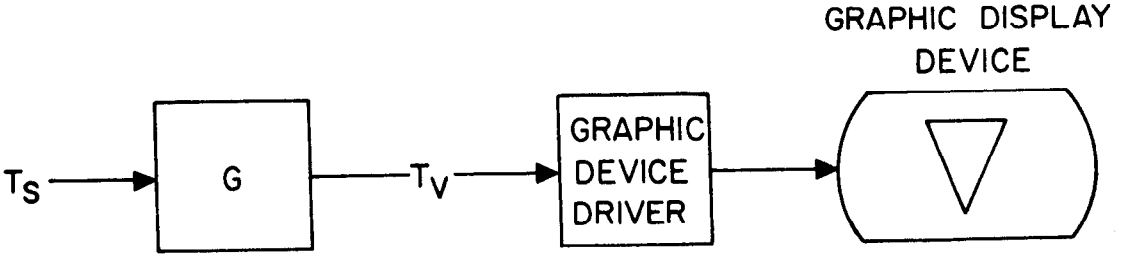


FIG. 3.3--Generation example.

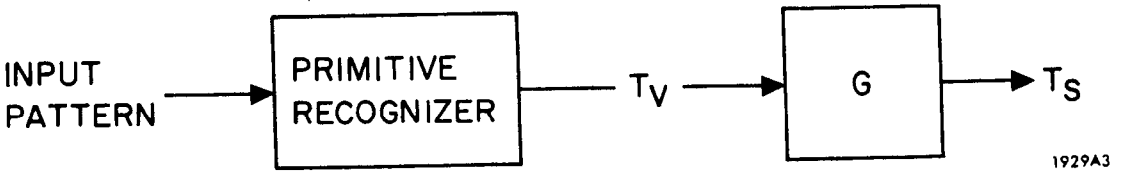


FIG. 3.4--Recognition example.

commands necessary to control the display device. This driver must recognize the construction elements in the BDF (e.g., lines, arcs, surfaces); in some cases, it will have to simulate an element if the particular device does not have a particular constructional element. To change graphic display devices, only the graphic device driver needs to be changed.

Fig. 3.4 illustrates recognition in a more complete form. The primitive recognizer analyzes the input pattern or picture and generates a string description (T_V) of it. Then, G translates this T_V into a T_S, which specifies how the primitives are associated to form the input pattern. If properly designed, this T_S could then be used to generate a T_V and the internal representation for the picture or pattern.

3.4 The Display Template

One of the more tedious problems with interactive systems is setting up and maintaining the communication device; it is even more tedious for a regenerating type display such as many of today's graphical displays. Further, many interactive systems are only available in the interactive mode and can easily become restricted to particular communication devices. This provided an early interest in providing device independence and ease of communication device usage within GEMS.

The interactive control program produced by GEMS from a user's control description assumes that two devices exist; an input communication device and an output communication device. Device independence is obtained by executing standard calls to user supplied routines to perform the communication input/output. The user supplies routines to initialize the input and output device (e.g., opening, attaching or assigning buffer storage to the device), to perform

input from the input device and to perform output to the output device; more detailed requirements for these routines are given in Section 4.1.

The output communication device is assumed to be a regenerating type display driven from a continguous buffer area; if this is not the case, the system still functions, but some unnecessary operations are performed. It is assumed that device instructions and/or data is written into this buffer area and that the display is continuously generated from this buffer. Thus, information continues being displayed until the corresponding data or instructions in the buffer area are replaced, modified or removed.

GEMS assumes that a pointer (BUFADD) to the last area of the display area used (or equivalently, the next open area) is maintained and used by the user's output communication routines to enter information into the display buffer and that this pointer is updated by this output routine. Further, GEMS divides the display buffer into eight logical areas (numbered 1 thru 8) and keeps a pointer to each of these areas; some or all may actually exist at any given time.

To replace or create an area, a procedure is called with an area number as a parameter (See Section 4.4 ---INITIALIZE procedure); this resets the last buffer area pointer (BUFADD) to this different area and resets all area pointers with a greater area number to zero (i.e., erases these areas). Thus, there is a message structure to the communication output device and the use of this structure by GEMS applications is designed to reduce the buffer overflow problem by providing these logical areas and the erasures implied by them.

The control description displays function and construction information in area 1, primitive names may be displayed in area 2, the top of the result stack may be displayed in area 3, the main action request in area 7 (i.e., the main

loop of the control procedure which is selecting a function or construction) and data for functions or constructions in area 8. Areas 4, 5 and 6 may be used by user programs and will not be erased by action requests. Several example templates are given in Chapter 5.

CHAPTER 4

IMPLEMENTATION OF THE MODEL

The model for graphical systems of Chapter 3 is implemented in PL/I and is presently used on an IBM 360/91. The model is implemented as three language pre-processors using SIMPLE (George 1971b)† and a common procedure library for accessing the data structure. The three language pre-processors are the control language (L_C), the definition language (L_D) and the graphic description language (L_G).

A typical application will consist of a control description (C), a definition description (D) and a graphic description (G). The control description (C) is translated into a control procedure by the control language processor (L_C); this control procedure is the executive program of an application and is the highest level program for an interactive application. The definition description (D) is translated into a definition procedure by the definition language processor (L_D). The graphic description language (L_G) translates the graphic description (G) into a parser and an associated semantic procedure.

For the typical application, these procedures are combined into the structure shown in Fig. 4.1 to form the desired system. Some example applications are given in Chapter 5.

4.1 The Control Language L_C

The control language (L_C) provides the mechanisms for describing the control executive for an application. Menu items and the sequence of operations to perform when the item is selected can be specified; additional support for specifying arguments for a menu item is provided. Procedures for updating the display are also provided.

† A brief explanation of SIMPLE is given in Appendix H.

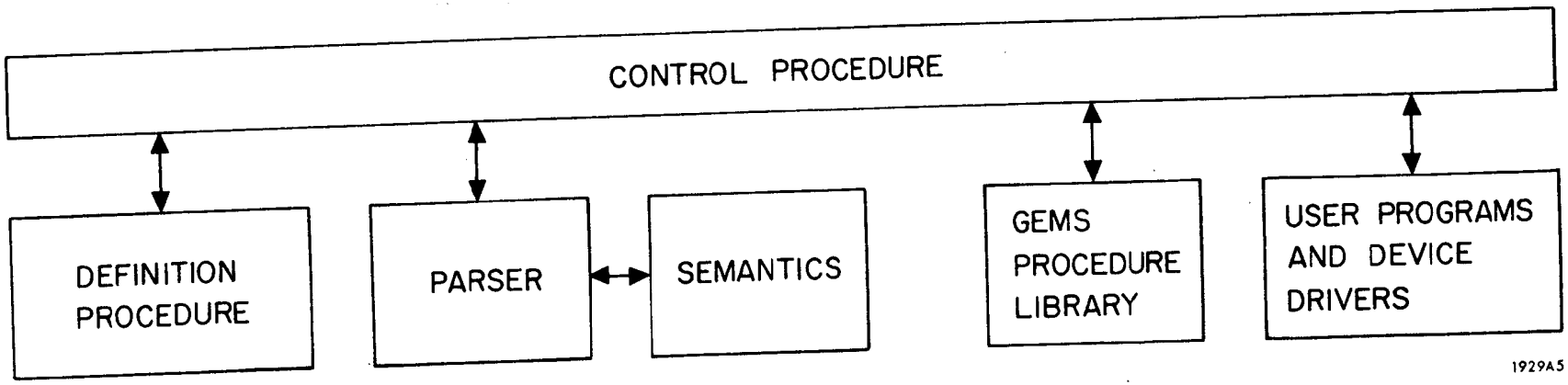


FIG. 4. 1--GEMS structure template.

The control language processor translates descriptions written in the control language (L_C) and PL/I into a PL/I procedure. This control procedure is the executive for an interactive application; an alternate entry point can be used for the slave mode.

In the interactive mode, the control procedure requests a keyword; the interpretation of the keyword is to request the specified operands or arguments and then to execute the semantics which are specified in PL/I. After the semantics have been executed, another keyword is requested.

In the slave mode, a string is passed containing all the keywords and data needed for the required action. For the initialization of variables, the keyword *INIT* is used.

The grammar for the control language is: †

Prod. No.		Production
1	CON_LANG	::= CONTROLA *CODE* ⟨PL/I⟩ *END* DESCRIPTION *END-CONTROL*
2	CONTROLA	::= *CONTROL* PROMPT DATA UNARY
3	PROMPT	::= STRING (SIMPLE terminal class)
4	UNARY	::= GROUP
5	GROUP	::= WORD (SIMPLE terminal class)
6		::= GROUP WORD
7	DATA	::= DATUM
8	DATUM	::= ITEM
9		::= DATUM ITEM
10	ITEM	::= SEP *=* STRING

† The entire input to SIMPLE, which defines L_C, is given in Appendix A. The grammar given here is a simple precedence grammar and contains some artificial productions (i.e., productions with no semantics).

```
11                          ::= QUOTES          *=*   STRING

12                          ::= CONNECTOR       *=*   STRING

13                          ::= CSEP            *=*   STRING

14                          ::= LEFT_PAREN      *=*   STRING

15                          ::= RIGHT_PAREN     *=*   STRING

16                          ::= TERMINATOR      *=*   STRING

17                          ::= MARKER          *=*   STRING

18                          ::= ASEP            *=*   STRING

19                          ::= MAX_PRIM        *=*   INTEGER (SIMPLE
                                                            terminal class)
20                          ::= LENGTH_PRIM     *=*   INTEGER

21  DESCRIPTION             ::= FUNCTION-PART

22                          ::= FUNCTION-PART   CONSTRUCTION-PART

23  FUNCTION-PART           ::= FUNA   OP-COMMANDS   *END*

24  FUNA                    ::= *FUNCTION*   STRING (STRING is function
                                                            class name)

25  CONSTRUCTION-PART       ::= CONA   OP-COMMANDS  *END*

26  CONA                    ::= *CONSTRUCTION*   STRING   (STRING is
                                                  construction class name)

27  OP-COMMANDS             ::= OPCOM

28  OPCOM                   ::= OP-COMMAND

29                          ::= OPCOM   OP-COMMAND

30  OP-COMMAND              ::= OPCON   OPERANDS *CODE* <PL/I>  *END*

31  OPCON                   ::= *NAME*  *=*  WORD  (WORD represents the
                                                  menu item or keyword)

32  OPERANDS                ::= OPERA

33  OPERA                   ::= *NONE*

34                          ::= OPERAND
```

35		::=	OPERA OPERAND
36	OPERAND	::=	STRING TYPE-OP (STRING is the prompt when asking for the operand)
37		::=	STRING TYPE-OP */* TYPE-OP
38	TYPE-OP	::=	CONTROL CLASS TYPE
39	CONTROL	::=	INTEGER
40		::=	* #*
41	CLASS	::=	PRIMITIVE
42		::=	RESULT
43	TYPE	::=	TYPE_A
44		::=	TYPE_B
45		::=	TYPE_C
46		::=	TYPE_D
47		::=	TYPE_E
48		::=	TYPE_F
49		::=	TYPE_G

The associated semantics in a functional form are (See Appendix A for the actual semantics):

Production	Semantics
1	Issue slave return or interactive loop transfer; issue the entry points and code for SAVE_PRIMITIVE, INPUTT, OUTPUTT, DISPLAY_PRIMITIVES, DISPLAY_RESULT and FIND_PRIMITIVE: issue control procedure termination.
2	Issue control procedure declaration (named INTERACTIVE), variable declarations, procedure entry declarations,

Production	Semantics
	variable initialization procedure (VAR_INIT), operand fetching procedure (OPERAND); issue call to connect communication devices (CALL SETUPI, CALL SETUPO) and to initialize variables (CALL VAR_INIT); issue slave entry point (names SLAVES).
3	Initialize MAX_PRIM and LENGTH_PRIM to 1.
5	Build assignment statement for first unary operator.
6	Update assignment statement and counter for current unary operator.
8	Initialize variables to null or zero and insert current variable value.
9	Insert current variable value.
10 - 20	Save value stack in proper form.
21	Issue dummy CONST procedure and CALL FUNCTION.
22	Issue CALL FUNCTION and CALL CONSTRUCTION.
23	Issue end of FUNCTION procedure; issue FUNCT procedure which displays the functions defined.
24	Save function class name for FUNCT; issue beginning of FUNCTION procedure (FUNCTION processes all function requests).
25	Issue end of CONSTRUCTION procedure; issue CONST procedure which displays the constructions defined.
26	Save construction class name for CONST; issue beginning of CONSTRUCTION procedure (CONSTRUCTION processes all construction requests).

28	Issue end for this OP-COMMAND.
29	Issue end for this OP-COMMAND; Save operation name for FUNCT or CONST.
31	Build if test for this option.
36	Issue operand call.
37	Issue operand call.
38	Save value stack.

The grammar for the control language can be divided into three parts; the data initializations, the function descriptions and the construction descriptions.

The data section is used to initialize variables related to the primitive representation (Section 4.2), the primitive storage allocation, the input recognizer (principally for the slave mode) and the construction of strings in the graphic description. In addition, all the unary operators are listed in this data section and are saved for use in constructing operands. The variables to be initialized are:

NAME	TYPE	EXPLANATION
ASEP	CHAR (20) VARYING	Attribute separator for the primitive representation (Section 4.2).
CONNECTOR	CHAR (20) VARYING	Attribute value pair connector for the primitive representation (Section 4.2).
CSEP	CHAR (20) VARYING	Construction separator in the basic display file of the primitive representation (Section 4.2).
LEFT_PAREN	CHAR (20) VARYING	That string which functions as a left parenthesis in the graphic description (Section 4.3).

LENGTH_PRIM	----------	Maximum character length of any primitive; used to directly initialize primitive array and is not an external variable.
MARKER	CHAR(20) VARYING	Used in building operands for functions and constructions; usually space or null (see Table 4.1).
MAX_PRIM	FIXED BINARY	Maximum number of primitives.
QUOTES	CHAR(20) VARYING	Used by INPUTT procedure (Usually in slave mode) to fetch an input string including blanks.
RIGHT_PAREN	CHAR(20) VARYING	That string which functions as a right parenthesis in the graphic description (Section 4.3).
SEP	CHAR(20) VARYING	Separator for the name, T_S and T_V fields of the primitive representation (Section 4.2); also separates the attribute list from the basic display file within a T_V.
TERMINATOR	CHAR(20) VARYING	Termination control for the indefinite control in the definition description (Section 4.2).

All of these variables, except LENGTH_PRIM, are declared as external variables in the control program and may be referenced by the function and/or construction descriptions, as well as external procedures.

The function and construction descriptions define a sequence of operations which are to be performed when a keyword is selected; the keyword is specified by the word class (Production 31). For the interactive mode, a keyword is requested by displaying the prompt (Production 3). Then a number of operands ($\leq$ 5) may be requested with the prompt specified by the string class (Production 36 or 37); these operands are left as a string of characters in ARG(1)...ARG(5) in the order they are received.

Each operand or argument may be either a primitive name or the contents of the top element of the result stack (RES_STACK), possibly modified by parentheses, unary operators and MARKER symbols. Table 4.1 illustrates the form of the operand as defined by Production 36 thru 47; where LEFT_PAREN is "(", RIGHT_PAREN is ")", MARKER is "A", # represents any unary operator and X represents any primitive name or the contents of the top element of the result stack.

The semantics for a function or construction are specified in PL/I; the form is *CODE* followed by any string which is interpreted as PL/I code and terminated by *END*.

In addition to the previous variables, the following variables are declared in the control procedure:

NAME	TYPE	EXPLANATION
ARG(5)	EXT CHAR(2000) VARYING	Operands are left here, in order by the OPERAND procedure.
ATEMP	CHAR(2000) VARYING	Input string buffer when requesting input.

NAME	TYPE	EXPLANATION
I	FIXED BINARY	Used in alternate entry points.
J	FIXED BINARY	Used in alternate entry points.
NUM_UNARY	EXT FIXED BINARY	Number of unary operators.
PRIMITIVES (MAX_PRIM)	EXT CHAR (LENGTH_PRIM) VARYING	Used for saving primitives.
RES_STACK(10)	EXT CHAR(200) VARYING	Used as a push down stack to save expressions of the graphic description; manipulated by the procedures POP and UPDATE.
SLAVE	EXT BIT(1)	True if program is in slave mode; false if in interactive mode.
TEMP	EXT CHAR(2000) VARYING	Input buffer when used in slave mode.

The control procedure is recursive and performs various functions; the entry parameters for the control procedure are IN, A, and B (all CHAR(*) VARYING). These entry points and there functions are:

NAME(PARAMETERS)	EXPLANATION
DISPLAY_PRIMITIVES(B)	Displays the primitive names in PRIMITIVES(*) with title B using procedures FETCH_NAME (Section 4.4) and OUTPUTT.

NAME(PARAMETERS)	EXPLANATION
DISPLAY_RESULT(B)	Displays the top of the result stack (RES_STACK(10)) using OUTPUTT.
FIND_PRIMITIVE(A,B)	Returns in B the primitive whose name is in A; null if no primitive by that name.
INPUTT(B)	If slave mode then returns next element of TEMP in B; if interactive then calls external user supplied procedure INPUT to return an input in B.
INTERACTIVE	Main entry point (interactive mode)
OUTPUTT(B)	If slave mode then dummy procedure; if interactive mode then call external user supplied procedure OUTPUTT to output B.
SAVE_PRIMITIVE(A)	Replaces or creates the primitive in A.
SLAVES (IN)	Slave mode entry point; decodes and performs actions specified in IN.

The following are internal procedures to the control procedure INTERACTIVE; the execution of these procedures is inherent in the control procedure.

NAME	EXPLANATION
CONST	Displays the construction class name and the keywords defined in the construction description.
CONSTRUCTION	Compares the input request to the construction keywords; if a match occurs, then the associated semantics are executed.

NAME	EXPLANATION
FUNCT	Displays the function class name and the function keywords defined.
FUNCTION	Compares the input request to the function keywords; if a match occurs, then the associated semantics are executed.
OPERAND(I, MSG, A, B)	Requests an operand using message MSG, requires it to be of type A or B and leaves the operand in ARG(I).
VAR_INIT	Variable initialization procedure.

Labels LO and EXIT1 are always used; the labels L1, L2, ... are used for constructions and functions defined.

The control program expects the following external procedures to be supplied by a user:

NAME (PARAMETER TYPE)	EXPLANATION
INPUT(CHAR(*) VARYING)	Obtain an input string from a device.
OUTPUT(CHAR(*) VARYING)	Output a string on a device. Update BUFADD.
SETUPI	Perform the initialization needed to connect the input device.
SETUPO	Perform the initialization needed to connect the output device; initialize BUFPTRS array.

SETUPO should declare BUFADD and BUFPTRS(*) (an array of 8 elements) as external fixed binary variables and initialize these to zero; the procedure INITIALIZE manipulates these variables (Section 3.4 and 4.4). BUFADD is used by OUTPUT as the last location in the buffer which has been used.

All of the library routines (See Section 4.4) are available from the control procedure.

Example 4.1 illustrates the use of the control language; together with Examples 4.2 and 4.3, these illustrate the use of GEMS.

EXAMPLE 4.1

For an example control description, consider a system with the following:

FUNCTIONS

EVAL (call an external procedure)

DEFINE (call the define procedure, update primitive display, redo result display)

CONSTRUCTIONS

+ (e.g., A+B)

(null meaning)

UNARY

(e.g., #A)

Some legal expressions are:

A + A

A + A etc.

(See Example 4.3 for the definition of the syntax and semantics for these expressions.)

The control description is:

```
*CONTROL*  'SELECT FUNCTION OR CONSTRUCTION'

    MAX_PRIM *=* 20   CSEP *=* ':'   CONNECTOR *=* '='  SEP *=* ';'
    ASEP   *=*  ' '  LENGTH_PRIM *=* 100
    #                                                   (unary operator)
```

```
        *CODE*
            DCL (EVAL, DEFINE) EXTERNAL ENTRY;
            *END*
*FUNCTION*  'FUNCTIONS'
        *NAME* *=* EVAL *NONE*
            *CODE*
                CALL EVAL;
                *END*
        *NAME* *=* DEFINE *NONE
            *CODE*
                CALL DEFINE;
                CALL DISPLAY_PRIMITIVES('PRIMITIVES ');
                CALL DISPLAY_RESULT('RESULT = ');
                *END*
            *END*
 *CONSTRUCTION*  'CONSTRUCTIONS'
        *NAME*  *=* + 'SELECT LEFT OPERAND'
            1 PRIMITIVE TYPE_A  */*  0 RESULT TYPE_A
            'SELECT RIGHT OPERAND'
            1 PRIMITIVE TYPE_A */*  0 RESULT TYPE_A
            *CODE*
                ARG(1) = ARG(1) || '+' || ARG(2);
                CALL UPDATE (ARG(1));
                CALL DISPLAY_RESULT('RESULT = ');
                *END*
```

```
    *NAME*   *=*    #    *NONE*
        *CODE*     *END*
    *END*
*END-CONTROL*
```

An example display generated by this control description is given in Fig. 4.2. Whenever EVAL or DEFINE is selected, then the appropriate action is taken. If + is selected, two operands are requested and then the top of the result stack is updated and displayed.

4.2 The Definition Language L_D and the Primitive Representation

The definition language (L_D) provides the mechanisms for describing how a specific primitive may be specified within the primitive representation. The goal of L_D is to provide the facility for describing an interactive procedure without specifying a communication device and to provide a reasonable level of checking upon the data requested.

In operation the definition procedure requests data using prompts defined by the user and checks the data for proper type. Using this data and keywords defined by the user, a primitive string is incrementally accumulated and finally saved in the primitive storage area.

4.2.1 The Primitive Representation

The data structure used for representing primitives and pictures is a modified version of the general form of Shaw (Shaw 1968a; Miller and Shaw 1967). A primitive or picture is represented by a string of three components: a unique primitive or picture name, a terminal syntactic string T_S and a terminal semantic string T_V. A basic primitive has a null T_S and a picture has a non-null T_S. Formally, the primitive representation is defined by:

```
FUNCTIONS  EVAL  DEFINE
CONSTRUCTIONS  +  #
PRIMITIVES   A  B
RESULT = A+#B
SELECT  FUNCTION  OR  CONSTRUCTION
```

1929A6

FIG. 4.2--Display for Example 4.1.

⟨PRIMITIVE⟩ ::= ⟨NAME⟩ ⟨SEP⟩ ⟨TS⟩ ⟨SEP⟩ ⟨TV⟩

⟨NAME⟩ ::= ⟨ID⟩ | ⟨INTEGER⟩

⟨ID⟩ ::= ⟨LETTER⟩ | ⟨ID⟩ ⟨LETTER⟩ | ⟨ID⟩ ⟨DIGIT⟩

⟨SEP⟩ ::= Any string not contained in any ⟨NAME⟩ , ⟨TS⟩ or ⟨TV⟩ except explicitly in the definition of ⟨TV⟩.

⟨TS⟩ ::= A terminal syntactic string of the desired graphic description; may be null.

⟨TV⟩ † ::= ⟨ATTR-LIST⟩ ⟨SEP⟩ ⟨BDF-FILE⟩

⟨ATTR-LIST⟩ ::= ⟨NULL⟩ | ⟨ATTRIBUTES⟩

⟨ATTRIBUTES⟩ ::= ⟨ATTRIBUTE⟩ | ⟨ATTRIBUTES⟩ ⟨ASEP⟩ ⟨ATTRIBUTE⟩

⟨ATTRIBUTE⟩ ::= ⟨NAME⟩ ⟨CONNECTOR⟩ ⟨VALUE⟩

⟨ASEP⟩ ::= A string not contained in any ⟨NAME⟩ , ⟨VALUE⟩ , ⟨CONNECTOR⟩ or ⟨SEP⟩ .

⟨CONNECTOR⟩ ::= Any string not contained in any ⟨NAME⟩ , ⟨VALUE⟩ , ⟨SEP⟩ or ⟨ASEP⟩ .

⟨VALUE⟩ ::= NUMBER | TEXT | NAME

⟨BDF-FILE⟩ ::= ⟨NULL⟩ | ⟨CONSTRUCTIONS⟩

⟨CONSTRUCTIONS⟩ ::= ⟨CONSTRUCTION⟩ | ⟨CONSTRUCTIONS⟩ ⟨CSEP⟩ ⟨CONSTRUCTION⟩

⟨CONSTRUCTION⟩ ::= ⟨NAME⟩ | ⟨NAME⟩ ⟨PARM-LIST⟩

⟨PARM-LIST⟩ ::= ⟨VALUE⟩ | ⟨PARM-LIST⟩ ⟨VALUE⟩

⟨CSEP⟩ ::= Any string except a leading string of any ⟨NAME⟩ or a trailing string of any ⟨NAME⟩ or ⟨VALUE⟩.

---

† ⟨TV⟩ is a terminal semantic string of the desired graphic description.

The ⟨ATTR-LIST⟩ contains the elements needed for associating primitives, such as the specification of tails and heads in the Picture Description Language (Shaw 1968a; Miller and Shaw 1967) as name value pairs. This design of the ⟨ATTR-LIST⟩ was intended to provide for multiple tail/heads, a varying number of tail/heads and to give some help for relational operators.

The basic display file (⟨BDF-FILE⟩) provides the graphic construction facilities (e.g., lines, curves, etc.) using a name and parameter list convention. With proper user design, several basic display files can be concatenated, using a separating string, to function as a composite string description of a picture; this string can then be decoded by a graphic driver program to generate a graphic display.

Although the formal structure of the basic display file is specified, the actual capabilities are specified by the user in his definition description and in his graphic description (Section 4.3). For each device to be used for graphic display, a driver must be supplied; by changing the driver program, the user can display his basic display files on various devices. Some of the capabilities used in the definition description for the graphic description may not be available on some devices, however the user has the option of ignoring or simulating these in each driver program.

A flexible basic display file should provide capabilities for the creation and placement of graphical symbols, for intensity and for simple transformations. The graphical symbols could include points, lines, curves and alphanumerics. A sophisticated alphanumeric capability would provide for character size and vertical or horizontal writing. The intensity should provide for multilevel intensity information, blinking and possibly color. The simple transformation should include translation, rotation and scaling; for dynamic displays, a time parameter could be used.

4.2.2 The Definition Language L_D

The definition language processor translates descriptions written in the definition language (L_D) into a PL/I procedure. This definition procedure has two modes of operation: interactive and slave.

The interactive mode allows the specified fields to be defined by providing prompting for the data, checking the input for errors and constructing a proper form of the primitive using various connectors and separators of the primitive representation.

The slave mode accepts a string as a valid primitive without prompting or checking. Both modes eventually save the primitive.

For the prompting, the user specifies communication utilizing the same device or devices used in the control procedure. The user is not burdened by maintenance of these devices in his definition specification; he simply specifies the messages to be sent as strings.

The grammar for the definition language is: †

Prod. No.			Production ††
1	DEF-LANG	::=	*DEFINITION* PROG_NAME DL *END-DEFINITION*
2	PROG_NAME	::=	WORD
3	DL	::=	DLA SPEC1 SPEC2 SPEC3
4	DLA	::=	*NAME* STRING
5 - 6	SPEC1	::=	*TS* | *TS* STRING
7 - 8	SPEC2	::=	*ATTR-LIST* | SPEC2A SPECA-
9	SPEC2A	::=	*ATTR-LIST* STRING
10	SPECA-	::=	SPECA

† The entire input to SIMPLE, which defines L_D is given in Appendix B.

†† WORD, INTEGER and STRING are SIMPLE terminal symbols.

11 - 12	SPECA	::=	SPEC | SPECA SPEC
13	SPEC	::=	WORD = NAME
14		::=	WORD = TEXT
15		::=	WORD = NUMBER
16 - 17	SPEC3	::=	*BDF-FILE* | SPEC3A SPECB-
18	SPEC3A	::=	*BDF-FILE* STRING
19	SPECB-	::=	SPECB
20	SPECB	::=	CONSTRUCTION
21		::=	SPECB *END* CONSTRUCTION
22 - 23	CONSTRUCTION	::=	CONST PARM-LIS | CONSTA PARM-LIS
24	CONST	::=	WORD INTEGER
25	CONSTA	::=	WORD #
26	PARM-LIS	::=	PARM-LIST
27 - 28	PARM-LIST	::=	PARM-PAIR | PARM-LIST PARM-PAIR
29	PARM-PAIR	::=	STRING NAME
30		::=	STRING TEXT
31		::=	STRING NUMBER

The associated semantics in a functional form are (See Appendix B for the actual semantic input to SIMPLE):

Production	Semantics
1	Output end of definition procedure.
2	Output name of procedure (PROG_NAME), procedure declaration, variable declarations, internal procedures for

checking name and number type, slave return and code for displaying procedure name upon interactive entry.

4 Issue code for requesting primitive name (using STRING as prompt), obtaining, checking and initialize building of the primitive.

5 Issue code to concatenate SEP to primitive since T_S field is to be null.

6 Issue code to request T_S (using STRING as prompt) and concatenating it to primitive.

7 Issue code to concatenate SEP to primitive since ATTR-LIST field is to be null.

8 End of non-null ATTR LIST; issue code to concatenate SEP to primitive, thereby ending the ATTR-LIST field.

9 Beginning of ATTR-LIST; issue code to display a message (using STRING) which will remain until end of ATTR-LIST.

13 Issue code to request an attribute (using WORD), check for type NAME, keep requesting until type NAME and concatenate WORD, CONNECTOR, data and ASEP to primitive.

14 Issue code to request an attribute (using WORD), accept any input as type TEXT and concatenate WORD, CONNECTOR, input data and ASEP to primitive.

15 Issue code to request an attribute (using WORD), check for type NUMBER, keep requesting until type NUMBER and concatenate WORD, CONNECTOR, input data and ASEP to primitive.

17 End of BDF-FILE; issue code to loop on construction type; output OPTION procedure which displays the construction options.

18 Beginning of BDF-FILE; issue BDF_FILE label and code to display a message (using STRING) which will remain until end of BDF-FILE.

21 Save construction options for OPTION procedure.

22 End of integer construction option; issue code to save primitive and exit definition procedure.

23 End of indefinite repeating option; issue appropriate ends.

24 Start of integer construction option type; WORD is type construction; issue code to check type and when successful to append type to primitive; issue loop on integer.

25 Start of indefinite repeating construction option; WORD is type construction; issue code to check type and when successful to append type to primitive; issue indefinite loop until TERMINATOR is received in response to "CONTINUE"; save primitive when TERMINATOR is received and then exit definition procedure.

29 Issue code to request (using STRING) name type parameter, check for type NAME, keep requesting until type NAME and when successful append a blank and the data to the primitive.

30 Issue code to request (using STRING) TEXT type parameter, accept any input and append a blank and data to the primitive.

31 Issue code to request (using STRING) NUMBER type parameter, check for type NUMBER, keep requesting until type NUMBER and when successful append a blank and then the data to the primitive.

To utilize the definition language to construct a definition procedure, the user must define five items; the program name, the prompt for the name field

of the primitive, the prompt for the T_S field if desired, the prompt and elements of the attribute list and the prompt and the elements of the basic display file.

The types TEXT, NUMBER and NAME are defined by:

TEXT = any character string.

NUMBER = any character string which does not raise the conversion, overflow or underflow condition in PL/I when assigned to a floating binary variable of default precision.

NAME = IDENTIFIER | INTEGER

IDENTIFIER = LETTER | IDENTIFIER LETTER | IDENTIFIER DIGIT

For communication, the definition procedure assumes the external procedures INPUT and OUTPUT discussed in Section 4.1.

EXAMPLE 4.2

We will define a primitive for examples 4.1 and 4.3. A primitive will be a line or two end points with two distinguished points (tail and head) and be limited to one dimension. The definition description is:

```
*DEFINITION*   DEFINE
        *NAME*  'ENTER PRIMITIVE NAME'
        *TS*     'ENTER TS'
        *ATTR-LIST*       'ENTER ATTRIBUTES'
              TAIL  =  NUMBER
              HEAD = NUMBER
        *BDF-FILE*      'SELECT CONSTRUCTION'
              LINE   1   'POINT1'  NUMBER  'POINT2'  NUMBER  *END*
```

POINT 1 'POINT1' NUMBER 'POINT2' NUMBER

END-DEFINITION

Using the specifications for the separators given in Example 4.1, an example primitive is

A;;TAIL=0 HEAD=1; LINE 0 1

4.3 The Graphic Language L_G

The graphic description specifies in L_G the linear string graphic application of interest. The graphic language processor translates the syntactic component of the description into a parser and the semantic component into a compatible semantic procedure.

L_G is designed, using the model of Chapter 3, assuming that the graphic description specifies how to determine the missing component of the terminal string T which represents a picture or a primitive (less the primitive name). Further, it is primarily designed to determine the semantic string T_V from T_S (i.e., the primary interest is generation), although it can be used to determine a T_S for some T_V's.

For a generation application, G_S specifies how a terminal syntactic string T_S is to be parsed for evaluation. † G_V specifies, for each parsing step, the operations to be executed. For most applications, these will involve manipulation of T_V; for example, the construction of transformations applied to the basic display file and related to information in the attribute list. It may also involve the manipulation of the attribute list. Thus, the model assumes that a composed picture can be constructed, step by step, by manipulating T_V.

† By evaluation is meant the determination of the unknown string (in this case T_V) from the known string (in this case T_S).

The graphic language processor consists of a syntax processor and an associated semantic definition facility. Both the syntax component and the semantic component may be utilized separately, thereby permitting a variable number of parsers and semantic procedures to exist in an application.

4.3.1 The Graphic Syntax Processor

The syntax processor is the syntax analyzer of SIMPLE with the executive of SIMPLE (George 1971b). The input requirements are discussed in the report; † those items relating to the semantic constructor of SIMPLE are to be ignored since this semantic constructor is not used (in actual use, the semantic constructor is not available to the linkage editing operation; this results in termination when it is called by the executive).

The skeleton parser †† used by the syntax analyzer is a modified version of that used in SIMPLE. The modifications are:

1. The parser is a callable procedure (not a main procedure) and thus has the following entry variables, in order;

   A. SINPUT CHAR(2000) VARYING

      The input string to be parsed.

   B. SOUTPUT CHAR(2000) VARYING

      The output string to be returned.

   C. SDIAG CHAR (2000) VARYING

      The diagnostic string to be returned.

   D. SEMANT ENTRY

      The name of the semantic procedure to be used.

2. The parsing and value stack sizes are decreased to an upper limit of 25 elements; the value stack reduced to CHAR(10) VARYING.

---

† The input requirements are also given in Appendix H.
†† The parser is given in Appendix C.

3. SCAN2, STACKOK and ERROR_RECOVERY procedures are deleted; no alternate scan of the input string, error diagnostics or error recovery mechanisms.

4. The modification discussed in SIMPLE (George 1971b, Sec. 5.2) is implemented.

5. The three diagnostic messages returned by the parser in SDIAG are;

   A. STACK OVERFLOW (i.e., more than 26 elements were used)

   B. PARSING ERROR (incorrect input string)

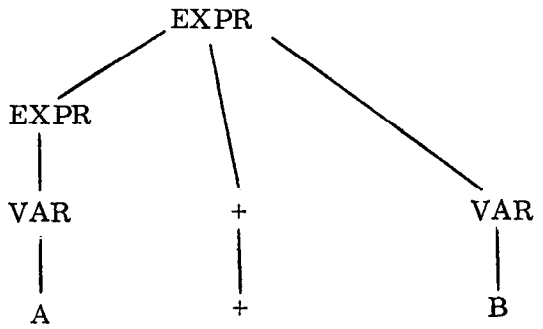
   C. INPUT OVERFLOW (desired stack manipulation by the semantic procedure would result in input string truncation)

The parser produced by the syntax processor assumes that the entire string to be parsed is given in a parameter upon initial entry and that this string is a valid string of the syntax. The parser internally concatenates the symbol 'TERMINAL' to the end of this input string. This resultant input string is then parsed until the TERMINAL symbol is encountered with the appropriate calls at each parsing step upon the indicated semantic procedure; the output and diagnostic strings are then assigned and control is returned to the calling procedure.

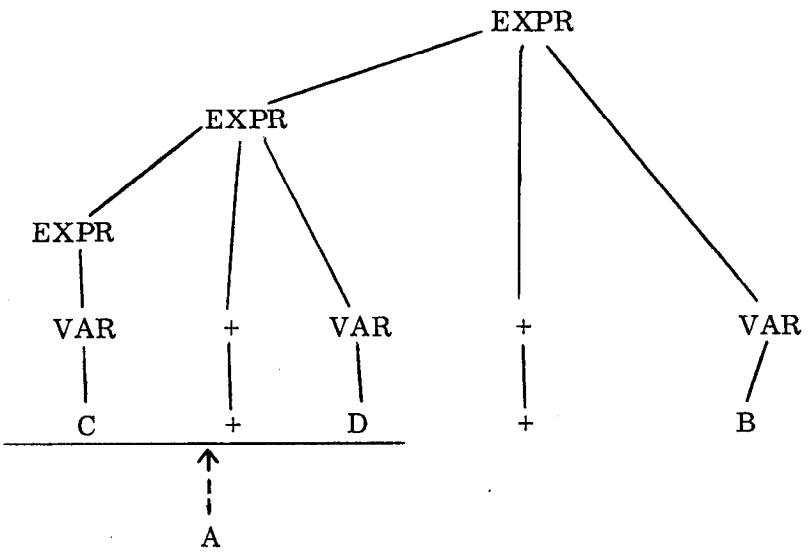
The parser produced also allows the parsing stack to be modified by the semantic procedure as discussed in the report (George 1971b, Sec. 5.2). As an illustration of the modification, consider the following grammar:

```
EXPR  ::= VAR
      ::= EXPR + VAR
VAR   ::= WORD
```

Further, assume that the string A + B is to be parsed. For normal parsing, the parse tree would be



Now assume the same input string, but that the semantics can determine that A is defined in terms of C and D. What is desired, is to replace A by C + D in the string A + B during parsing. The parse tree would be



Where the dotted line indicates that A was replaced by the string C +D, due to semantic action.

This modified parsing is accomplished in the parser by checking a switch (DIDDLE for please diddle the parsing stack); if the switch is true then the

current reducible substring is replaced by the contents of the string variable DIAG. The effect is as though the string substitution were originally made in the input string.

4.3.2 The Graphic Semantic Constructor

The semantic constructor is defined in SIMPLE (See Appendix C) and has the same input syntax as the semantic constructor of SIMPLE. The procedure constructed is compatible with the parser produced by the graphic syntax processor.

The procedure has the following parameters:

NAME - TYPE	EXPLANATION
N - FIXED BINARY	The production number for which the semantic reduction is required. This variable is used in selecting the proper section of the procedure to execute by the code generated by the semantic constructor.
VS - (0:25) CHAR(10) VARYING	The value stack; each element will initially contain a terminal symbol.
LEFT - FIXED BINARY	Points to the first element in the value stack corresponding to the left handle of the reducible substring.
RIGHT - FIXED BINARY	Points to the element of the value stack corresponding to the right handle (or end) of the reducible substring.
ANS - FIXED BINARY	For use in the semantics; initially set by the parser to zero.

NAME - TYPE	EXPLANATION
ERROR - BIT (1)	For use in the semantics; initially set by the parser to false.
OUT - CHAR(2000) VARYING	Output buffer; returned by the parser.
DIAG - CHAR(2000) VARYING	Diagnostic buffer; returned by the parser. Also used for stack modifications.
DIDDLE - BIT(1)	Used in communicating a stack modification or substitution to the parser by the semantics.

The semantic procedure also declares these internal variables:

NAME - TYPE	EXPLANATION
A - (20) CHAR(20) VARYING	Used for saving attribute values in string form.
I - (20) FIXED BINARY	Used for saving attribute values in integer form.
PTR - FIXED BINARY	Pointer to last used A, I and R; set to zero upon each entry into the semantic procedure.
R - (20) FLOAT BINARY	Used for saving attribute values in real number form.

The following internal procedures are defined in the semantic procedure:

NAME(PARAMETERS)	TYPE PROC.	EXPLANATION
DELETE(FIXED BINARY)	CHAR(20) VARYING	Converts its argument to a character string with no blanks.

NAME(PARAMETERS)	TYPE PROC.	EXPLANATION
FETCH(CHAR(2000) VARYING, CHAR(2000) VARYING)	--------	Fetches the values corresponding to the attribute names in the second argument from the primitive in the first argument; leaves string form in A(*), integer form in I(*) and real number form in R(*).
NEXT(CHAR(2000) VARYING)	CHAR(100) VARYING	Returns the next name in the argument after deleting it.
SAVE(CHAR(2000) VARYING, CHAR(2000) VARYING)	CHAR(2000) VARYING	Saves the values from A(*) with the names in the second argument into the attribute list in the first argument; A(PTR) is associated with the first name, A(PTR+1) with the second name, etc.

In addition, the library routines FETCH_ATTR, FETCH_BDF, FETCH_TS, FETCH_NAME, FETCH_PRIMITIVE, FETCH_VALUE and all the SAVE_XXX procedures are available from the semantic procedure (See Section 4.4); the variables CSEP, ASEP, LEFT_PAREN, RIGHT_PAREN, MARKER and CONNECTOR are also available (See Sections 4.1 and 4.2).

The graphic semantic procedure is organized to function utilizing the value stack to save pointers to partially modified data. For each production of the syntax for which semantic action is required, code written in PL/I must be provided. For every entry into the semantic procedure, the value stack and the left and right pointers are given; if temporary storage is needed to save partial results, then a pointer may be left in the value stack. Further, due to the parser action, all elements of the value stack to the right of the left pointer are effectively erased after return to the parser.

A short example will be used to illustrate the graphic semantic procedure.

EXAMPLE 4.3

Let the graphic syntax be:

PROD. NO.		PRODUCTION	
1	PICTURE	::=	SUB_PICTURE
2	SUB_PICTURE	::=	PRIMITIVE
3		::=	SUB_PICTURE + PRIMITIVE
4	PRIMITIVE	::=	WORD
5		::=	# PRIMITIVE

Further assume that all primitives have two distinguished points, namely a Tail and a Head, denoted by T and H. Also assume that the primitives are limited to one dimension.

Define + and # by:

Let A be $T \longrightarrow H$ and B be $T \longrightarrow H$

then,

$A + B = T \longrightarrow H$ (head to tail concatenation)

$\# A = H \longleftarrow T$

The translation for the + operator will be accomplished by a TRANS feature assumed defined in the basic display file.

The semantics needed for the productions are:

PROD. NO.	SEMANTICS
1	Output the SUB_PICTURE
2	No action
3	Perform the concatenation
4	Find the primitive; if its T_S field is blank then save a pointer to the primitive in the value stack; if the T_S field is not blank then substitute this for this occurrance of this primitive name in the input string.
5	Reverse the tail and head; move pointer to primitive to left element of stack.

The input to the graphic semantic constructor for a semantic procedure named EXAMPLE using an external array PIC for work area and ANS pointing to the last used PIC is:

```
*SEMANTICS*      EXAMPLE
    *CODE*
```

```
            DCL PIC(5) EXT CHAR(2000) VAR;
            *END*
*PRODUCTION*    1
    *CODE*
            OUT = PIC(VS(LEFT));
            *END*
*PRODUCTION*    3
    *CODE*
            CALL FETCH(PIC(VS(RIGHT)), 'TAIL HEAD');
            CALL FETCH(PIC(VS(LEFT)), 'HEAD');
            I(3) = I(3) - I(1);
            A(1) = I(2) + I(3);
            A(3) = DELETE(I(3));
            PTR = O;
            /* A(1) is the new head, A(3) is the translation */
            CALL SAVE_ATTR(PIC(VS(LEFT)),
                SAVE(FETCH_ATTR(PIC(VS(LEFT))), 'HEAD'));
            /* PIC(VS(LEFT)) now has the proper tail head */
            PIC(VS(RIGHT)) = FETCH_BDF(PIC(VS(LEFT))) ||
                CSEP || 'TRANS' || A(3) || CSEP ||
                        FETCH_BDF(PIC(VS(RIGHT)));
            /* PIC(VS(RIGHT)) now has the composite BDF file */
            CALL SAVE_BDF(PIC(VS(LEFT)), PIC(VS(RIGHT)));
            ANS = VS(LEFT); /* reset pointer to last used PIC */
            *END*
```

```
*PRODUCTION*  4
    *CODE*
            IF ANS < 5 THEN DO;
                ANS = ANS + 1;
                PIC(ANS) = FETCH_PRIMITIVE(VS(LEFT));
                VS(LEFT) = ANS;
                DIAG = FETCH_TS(PIC(ANS));
                IF DIAG = ' '  |  DIAG = ' '  THEN RETURN;
                    ELSE DO;
                        DIDDLE = '1'B;
                        ANS = ANS - 1;
                        RETURN;
                        END;
                    END;
            * END*
*PRODUCTION*  5
    *CODE*
            CALL FETCH(PIC(VS(RIGHT)), 'TAIL HEAD');
            PTR = O;
            CALL SAVE_ATTR(PIC(VS(RIGHT)), SAVE(FETCH_ATTR(PIC(VS
                (RIGHT))), 'HEAD TAIL'));
            VS(LEFT) = VS(RIGHT);
            *END*
*END-SEMANTICS*
```

4.4 GEMS Procedure Library

The procedures contained in the procedure library are:

NAME-PARAMETERS	TYPE	EXPLANATION
FETCH_ATTR - CHAR(*) VARYING	CHAR(2000) VARYING	Returns the attribute list from the primitive in the argument.
FETCH_BDF - CHAR(*) VARYING	CHAR(2000) VARYING	Returns the basic display file from the primitive in the argument.
FETCH_NAME - CHAR(*) VARYING	CHAR(2000) VARYING	Returns the name field from the primitive in the argument.
FETCH_PRIMITIVE - CHAR(*) VARYING	CHAR(2000) VARYING	Returns the primitive whose name is in the argument.
FETCH_TS - CHAR(*) VARYING	CHAR(2000) VARYING	Returns the T_S field from the primitive in the argument.
FETCH_VALUE - CHAR(*)VARYING, CHAR(*)VARYING	CHAR(2000) VARYING	Returns the value corresponding to the name in the second argument from the attribute list in the first argument.
INITIALIZE - FIXED BINARY	---------	Declares BUFADD and BUFPTRS(8) FIXED BINARY. If BUFFADD is less than BUFPTRS

NAME-PARAMETERS	TYPE	EXPLANATION
		(argument) then sets BUFPTRS(arg) to BUFADD; if BUFPTRS(arg) is zero then sets BUFPTRS(arg) and BUFFADD to the minimum of BUFPTRS (arg+1 to 8) and BUFFADD; else sets BUFADD to BUFPTRS(arg). In all cases BUFPTRS(arg+1 to 8) are set to zero. This is the maintenance of the buffer pointer and area organization discussed in Section 3.4.
POP	CHAR(2000)VARYING	Pops up the result stack, sets first element to null and returns the top element.
SAVE_ATTR - CHAR(*) VARYING, CHAR(*)VARYING	----------	Saves the attribute list in the second argument into the primitive in the first argument.

NAME-PARAMETERS	TYPE	EXPLANATION
SAVE_BDF -	--------	Saves the basic display file in the second argument into the primitive in the first argument.
CHAR(*) VARYING,		
CHAR(*) VARYING		
SAVE_TS -	--------	Saves the T_S field in the second argument into the primitive in the first argument.
CHAR(*) VARYING		
CHAR(*) VARYING		
SAVE_VALUE -	--------	Saves the value in the third argument with the name in the second argument into the attribute list in the first argument.
CHAR(*) VARYING,		
CHAR(*) VARYING,		
CHAR(*) VARYING		
UPDATE	---------	Pushes down the result stack and sets the top element to the argument.
CHAR(*) VARYING		

These procedures are given in Appendix D.

TABLE 4.1

NUMBER OF UNARY OPERATORS ALLOWED	TYPE_A	TYPE_B	TYPE_C	TYPE_D
0	X	(AXA)	(AXA)	(AXA)
1	#AX	#A(AXA)	(A#AXA)	(A#A(AXA)A)
2	#A#AX	#A#A(AXA)	(A#A#AXA)	(A#A(A#A(AXA)A)
indefinite	#A#A...#AX	#A#A...#A(AXA)	(#A#A...#AXA)	(A#A(A#A...(A#A

	TYPE_E	TYPE_F	
0	X	X	
1	(A AXA)	#A(AXA)	
2	(A A(A AXA)A)	#A(A#A(AXA)A)	(A#
indefinite	(A A(A A...(A A(A AXA)A)...A)	#A(A#A(A...(A#A(AXA)A)...A)	(A#A(A#A.

where,

LEFT_PAREN = (RIGHT_PAREN =) MARKER = A UNARY = #

and X represents a primitive or the top of the result stack

CHAPTER 5

GEMS APPLICATIONS

5.1 Two-Dimensional Mathematical Expressions

Early interest in the recognition and generation of two-dimentional mathematical expressions resulted from the disparity between normal mathematical notation and the linear notation required as input to most scientific compilers or produced by early symbolic systems.

MADCAP (Wells 1961, 1963) was an early compiler which accepted two-dimensional expressions from typewriter devices; internally the expression was kept as a two-dimentional array--a replica of the expression as it appeared on a printed page. This was then analyzed and converted to a linear expression for use by the compiler. Klerer, May and Grossman (Klerer and May 1965; Klerer and Grossman 1967) also developed a two-dimensional programming system which was typewriter oriented. In this system, arbitrary size symbols could be constructed from elementary strokes; the two-dimensional input is analyzed and converted to a linear representation using a character array for all possible typewriter positions. Although originally designed to recognize mathematical expressions, the basic techniques were extended to manipulate and format mathematical text. More recently, several systems have been utilizing a RAND Tablet with projection for input and output (Anderson 1968; Bernstein and Williams 1968); both of these have the two-dimensional recognition capability but are not necessarily tied to a compiler--the input is analyzed and converted to an internal linear form which may be used by various subsystems. This internal form is converted to a two-dimensional form for user inspection and feedback.

Along with the interest in scientific compilers with a two-dimensional capability came an interest in two-dimensional output from symbolic manipulation by computer. Martin's Symbolic Mathematical Laboratory (Martin 1967) is implemented in LISP and utilizes a scope for user communication and a plotter for hard copy; the two-dimensional output is provided by a system implemented by Krakauer (Krakauer 1964). CHARYBDIS (Millen 1968) is a program to display the output from MATHLAB on typewriter like devices. In the previous symbolic manipulation systems, the input is linear and the output expressions have generally been given as two-dimensional expressions. A recent system (Blackwell and Anderson 1969) has two-dimensional input and output and manipulates the mathematical expressions by user supplied rules.

The conversion of a linear expression to a two-dimensional expression is thus an important concept for symbolic manipulation and for giving a user an indication of how his input is being recognized. For GEMS, this is an interesting problem in light of the following questions:

1. How will the syntax for parsing an expression to generate a two-dimensional display compare to one used for evaluation in an interpreter or for code generation in a compiler?
2. Can other useful semantics be associated with this diaplay syntax?
3. Does GEMS have the necessary flexibility and power for this application?
4. If this application is developed interactively, will it be useful in some non-interactive mode?

Consider the definition of a system to construct, display and evaluate simple linear mathematical expressions (i.e., those expressions containing simple variables and the mathematical operators; addition, subtraction,

multiplication, division and exponentiation). Using GEMS this is accomplished by the components; the control description, the definition description and the graphic description. In addition, device drivers will be needed to generate a graphical display from the graphical description generated.

5.1.1 The Control Description

The control description determines the user interface in both the interactive and slave modes. The functions to be defined are:

NAME	EXPLANATION
ASSIGN	Assigns the current expression to the T_S part of a primitive.
DEFINE	Allows new primitives to be defined.
DISPLAY	Generates a two-dimensional representation of the current expression on the scope.
EVAL	Evaluates the current expression and displays the result on the scope.
PRINT	Generates a two-dimensional representation of the current expression on a printer.
OUTPUT	Generates an internal representation of the current expression.
PRINTER	Outputs the internal representation to the printer.
INIT	Allows initialization of parameters.
INPUT	Allows a character string to be entered as the current expression.
POP	Resets the current expression to the previous expression.

NAME	EXPLANATION
TERM	Terminate the system.
DBG	Set debug mode. When in debug mode, more internal information is output in both the interactive and slave modes.

The constructions to be defined are:

+	Addition; select two operands and update the current expression.
-	Subtraction; similarily.
/	Division;
*	Multiplication;
**	Exponentiation;
NEG	Negate the current expression.

Each of the binary operators (+,-,/,* and **) allow one minus sign to appear before any operand; further, the proper parenthesization is returned as part of the operand.

With the display template, this specifies the interactive user interface (see Fig. 5.1 for example). The functions are listed as options and the constructions as operators. Whenever an option or operator is selected, the operands (if any are required) are requested and only after the operands are obtained, is the specified action taken. In the interactive mode, the system will request operands until they are received; in the slave mode, exit results from exhausting the current input string.

To construct the string A+B in the interactive mode (assuming A and B are defined primitives), the following steps would be required:

1. Enter + (either keyboard entry or light pen selection)

2. Enter A (either keyboard entry or light pen selection)

3. Enter B "

A+B would now appear as the current expression. An alternate method would be:

1. Enter INPUT

2. Enter A+B

For the slave mode, one would execute

CALL SLAVES('+ A B'); or CALL SLAVES('INPUT A+B');

The actual control description is given in Appendix E. The external communication procedures used by several applications are given in Appendix F.

5.1.2 The Definition Description

For this application, a primitive is a mathematical variable or constant; a primitive will thus consist of a name and a numerical value. This definition description is given in Appendix E.

5.1.3 The Graphic Description

The graphic description specifies how to parse T_S and what semantics to apply to obtain the desired T_V. It consists of a syntax description and two semantic descriptions; one for the two-dimensional display and one for normal arithmetic evaluation.

5.1.3.1 The Syntax Description

The strings to be parsed (T_S's) will consist of variable names, operators and parentheses. The binary operators are +, -, /, * and **. All are binary operators except '-' which may also be a unary operator. / and * have precedence over + and -; ** has precedence over /, *, + and -; parentheses can override the normal precedence of the operators and can determine the scope of the operators.

The syntax description for parsing a mathematical expression composed of these operators and variables is given in Appendix E. This syntax is a simplified form of an arithmetic expression in EULER (Wirth and Weber 1966a and b). Thus, this is a syntax for arithmetic evaluation of an expression. Now the question is 'What is the syntax to generate an equivalent two-dimensional display of the expression?'.

The arithmetic syntax indicates whenever any operator or set of parentheses occurs; this is precisely what is needed in order to generate a two-dimensional display of the expression. This is the origin of the term "evaluating a string description to obtain a picture" (George 1967c, 1968a and b, 1969 a and b; Shaw 1969b); the term evaluating was chosen because of the realization of the similarity between evaluating an arithmetic expression to obtain a numerical value and evaluating the same string to obtain a two-dimensional display of it. In this case, the only differences between the two are the semantic rules applied.

5.1.3.2 The Arithmetic Semantic Description

In addition to the normal mathematical operations, certain error conditions must be detected. These error conditions may be divided into three classes; mathematical, restrictions of the programming language in which the semantics are implemented and the design of the evaluation implementation. Whenever any of these conditions occur, the evaluation is terminated and a diagnostic message is generated.

The mathematical conditions are undefined variable, illegal or invalid expression and division by zero.

The error conditions related to restrictions of the programming language are a negative number raised to a power, conversion error, overflow and

underflow. Overflow (too large a result) and undeflow (too small a result) result from the data attributes used in the semantics; in this case float binary of default precision in PL/I. A conversion error can occur when the value of a variable is retrieved in string form and converted to arithmetic type. The negative number raised to a power results from the definition of real exponentiation in PL/I and is detected by the error condition which also raises the same condition for an argument of zero and the exponent less than or equal to zero.

The error conditions related to the design of the evaluation implementation are evaluation stack overflow and parsing stack overflow. The parsing stack overflow is detected by the parser (created from the syntax description) when a new stack element is needed; this can only be changed by minor changes in the graphic language of GEMS. The evaluation is accomplished by keeping partial results during parsing in a stack of the semantic routine and a pointer to this stack in the value stack of the parser. The size of this stack is under control of the semantic description.

For the semantics, partial results are stored in an array (REAL FLOAT BINARY) and the variable ANS(See Sec. 4.3) points to the last element used. Assignment to this array is made when a basic variable (one defined by value and not in terms of other variables) is recognized and a pointer to the value in the array is left in the value stack (VS, Sec. 4.3); upon recognition of a non-basic variable (i.e., one defined by an expression), the semantics substitutes the expression for the variable and returns to the parser. All arithmetic operations are performed between adjacent elements of the array; elements are released after a binary arithmatic operation. The arithmetic semantic description is given in Appendix E.

5.1.3.3 The Display Semantic Description

For display, the problem is identical from a parsing viewpoint, but different from the semantic viewpoint. First, it is not required that a variable be defined (i.e., have a value and be in the list of defined primitives), since its occurrence is sufficient definition for display purposes. Second, rather than to evaluate the expression for a single numerical value, a description from which a two-dimensional display can be generated must be created.

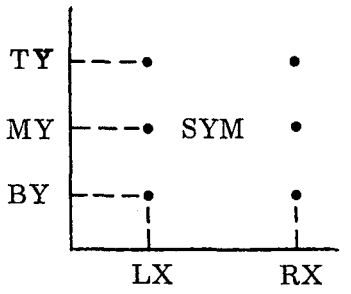
GEMS primitive representation will be used as the data representation for variables, intermediate expressions and for the target string description. The representation to be used is:

$$\langle SEP \rangle \ \langle SEP \rangle \langle TV \rangle$$

and,

$$\langle TV \rangle = \langle ATTR\text{-}LIST \rangle \ \langle SEP \rangle \ \langle BDF\text{-}FILE \rangle$$

The attribute list will contain information useful in generating the target description which is the $\langle BDF\text{-}FILE \rangle$. For recognizing two-dimensional mathematical expressions, six data points for symbols have been sufficient (Anderson 1968); although generation can be accomplished with less (Krakauer 1964; Martin 1967), it is sometimes necessary to generate virtual points. Hence, six points will be used for elementary symbols and are illustrated by:



where,

LX = minimum x coordinate (for elementary symbols zero)

RX = maximum x coordinate

BY = minimum y coordinate

TY = maximum y coordinate

MY = value between BY and TY; for elementary symbols it will be (TY+BY)/2 (i.e., zero)

For intermediate results, the type of the last operator and whether or not the expression is negated is needed to generate aesthetic displays with a minimum of parenthesization. This information is also contained in the attribute list by:

TYPE	= 0	variables or parenthesized expression
	= 1	addition last operator
	= 2	subtraction
	= 3	multiplication
	= 4	division
	= 5	exponentiation

and,

UNARY	= 0	No negation at highest level
	= 1	Expression negated

The ⟨BDF-FILE⟩ is designed using a keyword and parameters and is adequate for elementary symbols and expressions; the constructions for the ⟨BDF-FILE⟩ are:

ALPHA	⟨NAME⟩	for variables and operators
LINE	{ ⟨INTENSITY⟩ ⟨X⟩ ⟨Y⟩ }	division and brackets †

† { } means occurs zero or more times.

TRANS ⟨X⟩⟨Y⟩ for associating expressions

UNTRAN ⟨X⟩ ⟨Y⟩ "

Within the semantics, only binary or unary operations will involve graphical tasks. Hence, the semantics will be illustrated using

⟨LE⟩ ⟨OP⟩ ⟨RE⟩ for binary operators

and,

⟨OP⟩ ⟨RE⟩ for unary operators

where ⟨LE⟩ and ⟨RE⟩ represent the left and right expressions associated by operator ⟨OP⟩ .

Further, references to attributes of an expression will be indicated by attribute name and expression

e.g. $^{TX}\langle LE\rangle$

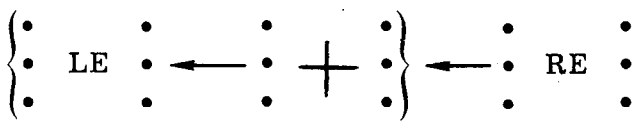
the ⟨BDF-FILE⟩ will be indicated by

$^{BDF}\langle LE\rangle$

The subscript R will imply the resultant intermediate expression of a semantic interpretation rule.

The semantics are:

<u>ADDITION</u> ---------- ⟨LE⟩ + ⟨RE⟩



IF $UNARY_{\langle RE \rangle} = 1$ THEN BRACKET $\langle RE \rangle$ †

TRANSLATE BDF_+ TO $(RX_{\langle LE \rangle}, MY_{\langle LE \rangle})$ ††

TRANSLATE $BDF_{\langle RE \rangle}$ TO $(RX_{\langle LE \rangle} + CHAR\_SIZE\_X, MY_{\langle LE \rangle})$

$BDF_R = BDF_{\langle LE \rangle} \langle CSEP \rangle BDF_+ \langle CSEP \rangle BDF_{\langle RE \rangle}$

$RX_R = RX_{\langle RE \rangle}$

TY_R = MAXIMUM of $TY_{\langle LE \rangle}$ and $TY_{\langle RE \rangle}$

BY_R = MINIMUM of $BY_{\langle LE \rangle}$ and $BY_{\langle RE \rangle}$

$MY_R = MY_{\langle LE \rangle}$

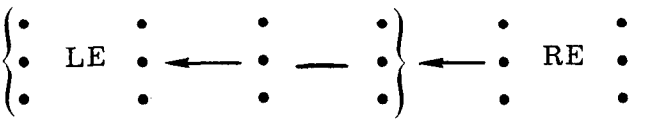
$LX_R = LX_{\langle LE \rangle}$

$TYPE_R = 1$

$UNARY_R = UNARY_{\langle LE \rangle}$

SUBTRACTION ---------- $\langle LE \rangle$ - $\langle RE \rangle$

same as addition except



IF $TYPE_{\langle RE \rangle} = 1$ or 2 or $UNARY_{\langle RE \rangle} = 1$ THEN BRACKET $\langle RE \rangle$

and,

$TYPE_R = 2$ (and - symbol)

---

† BRACKET $\langle -- \rangle$ means to draw brackets about the expression and set its type and unary to zero and modify its attribute list accordingly.

†† TRANSLATE $\langle -- \rangle$ to (X, Y) means to translate the BDF to (X, Y) and to modify the attribute list accordingly.

NEGATION ---------- - ⟨RE⟩

IF $TYPE_{\langle RE \rangle}$ = 1 or 2 or $UNARY_{\langle RE \rangle}$ = 1 THEN BRACKET ⟨RE⟩

TRANSLATE $BDF_{\langle RE \rangle}$ TO (RX_, MY_)

BDF_R = BDF_ ⟨CSEP⟩ $BDF_{\langle RE \rangle}$

LX_R = LX_

RX_R = $RX_{\langle RE \rangle}$

TY_R = $TY_{\langle RE \rangle}$

MY_R = $MY_{\langle RE \rangle}$

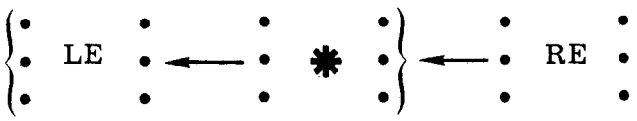
BY_R = $BY_{\langle RE \rangle}$

$TYPE_R$ = $TYPE_{\langle RE \rangle}$

$UNARY_R$ = 1

MULTIPLICATION ---------- ⟨LE⟩ * ⟨RE⟩

same as addition except

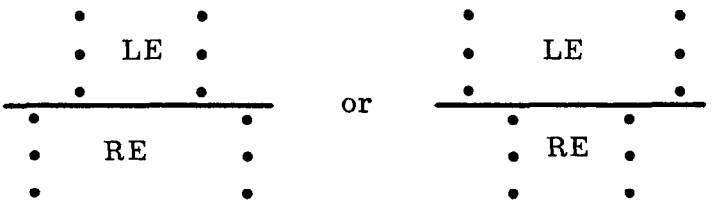


IF $TYPE_{\langle RE \rangle}$ = 1 or 2 or $UNARY_{\langle RE \rangle}$ = 1 THEN BRACKET ⟨RE⟩

IF $TYPE_{\langle LE \rangle}$ = 1 or 2 THEN BRACKET ⟨LE⟩

and, $TYPE_R$ = 3 (and * symbol)

DIVISION ---------- LE / RE



IF $TYPE_{\langle LE \rangle} = 4$ THEN BRACKET $\langle LE \rangle$

IF $TYPE_{\langle RE \rangle} = 4$ THEN BRACKET $\langle RE \rangle$

X1,X2 = 0 Calculate the x translations for numerator and denominator

IF $RE_{\langle LE \rangle} - LX_{\langle LE \rangle} < RX_{\langle RE \rangle} - LX_{\langle RE \rangle}$

$$\text{THEN } X1 = \frac{(RX_{\langle RE \rangle} - LX_{\langle RE \rangle} - (RX_{\langle LE \rangle} - LX_{\langle LE \rangle}))}{2}$$

$$\text{ELSE } X2 = \frac{(RX_{\langle LE \rangle} - LX_{\langle LE \rangle} - (RX_{\langle RE \rangle} - LX_{\langle RE \rangle}))}{2}$$

TRANSLATE $BDF_{\langle LE \rangle}$ TO (X1, CHAR_SIZE_Y / DIVS) †

TRANSLATE $BDF_{\langle RE \rangle}$ TO (X2,-CHAR_SIZE_Y / DIVS)

CREATE BDF_{LINE} a horizontal line from LUNDER to

$MAX(RX_{\langle LE \rangle}, RX_{\langle RE \rangle}) + RUNDER$ †

$BDF_R = BDF_{\langle LE \rangle} \langle CSEP \rangle BDF_{LINE} \langle CSEP \rangle BDF_{\langle RE \rangle}$

† These are related to device characteristics and will be discussed in the next section.

$LX_R = 0$

$RX_R = MAX(RX_{\langle LE \rangle}, RX_{\langle RE \rangle})$

$TY_R = TY_{\langle LE \rangle}$

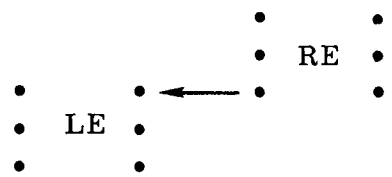
$BY_R = BY_{\langle RE \rangle}$

$MY_R = 0$

$TYPE_R = 4$

$UNARY_R = 0$

<u>EXPONENTIATION</u> ---------- $\langle LE \rangle$ ** $\langle RE \rangle$



IF $TYPE_{\langle LE \rangle} \neq 0$ or $UNARY_{\langle LE \rangle} = 1$ THEN BRACKET $\langle LE \rangle$

TRANSLATE $BDF_{\langle RE \rangle}$ TO $(RX_{\langle LE \rangle}, TY_{\langle LE \rangle})$

$BDF_R = BDF_{\langle LE \rangle} \langle CSEP \rangle \; BDF_{\langle RE \rangle}$

$RX_R = RX_{\langle RE \rangle}$

$LX_R = LX_{\langle LE \rangle}$

$TY_R = TY_{\langle RE \rangle}$

$MY_R = MY_{\langle LE \rangle}$

$BY_R = BY_{\langle LE \rangle}$

$TYPE_R = 5$

$UNARY_R = 0$

The actual display semantics are given in Appendix E.

5.1.4 Discussion of the Two-Dimensional Example

Although GEMS is designed to provide systems which are device independent, the two-dimensional mathematical display problem is highly dependent upon the characteristics of the display device used, as indicated by the appearance of various variables in the semantics. Although a ⟨BDF-FILE⟩ could have been created to handle any device, it is more convenient to arrange for the ⟨BDF-FILE⟩ being more related to the device. This simplifies the implementation of the driver for the device. The variables concerned are:

NAME	PRINTER	SCOPE	EXPLANATION
CHAR_SIZE_X	2	21	Character size in the x direction.
CHAR_SIZE_Y	1	30	Character size in the y direction.
CDIV	2	1	Division for CHAR_SIZE_X at basic symbol recognition time.
NSPACE	1	0	Space in units to be left at end of symbol.
DIVS	1	4	Division for CHAR_SIZE_Y for constructing the division line.
BWIDTH	1	10	Width for brackets.
BHIGH	1	0	Height for brackets.
LUNDER	0	-11	Units to right to begin division line.
RUNDER	-2	-15	Units to right to end division line.

A device driver is passed the string returned by the parser-semantics; it is the primitive representation used for intermediate expressions. The task of the device driver is the conversion of the ⟨BDF-FILE⟩ to whatever is necessary to create the desired output upon the device. Drivers for the printer and the scope (IBM 2250) are given in Appendix E.

Figures 5.1 and 5.2 illustrate the two-dimensional mathematics example in the interactive mode using an IBM 2250 Graphic Display. As illustrated, the system uses parentheses only where necessary; e.g.,

$$\left[[A+B]^{A+B} \right]^{[A+B]^{A+B}} * B \quad \text{in Fig. 5.1}$$

and

$$\frac{D}{C+D}_{B} \quad \text{in Fig. 5.2}$$

Both of these were choices taken in the definition of the display semantics. Figures 5.3 and 5.4 illustrate the printer output when used in the slave mode. Fig. 5.5 illustrates the use of the two-dimensional mathematical expression display generation system as a sub-task of a text printing program; Fig. 5.6 illustrates the interactive system running utilizing a typewriter.

There were no programming changes required in the two-dimensional math example in order to use the system, developed as an interactive application, as a slave program or as a procedure of the text printing program. Clearly, any system implemented via GEMS could be connected to this printing program; e.g., a drawing program for constructing figures, a flowcharting system to prepare flowcharts, etc.

5.2 A Drawing System

A drawing system was needed to produce a wide range of test input for pattern recognition experiments. An interactive system was preferred with sample output being displayed on a scope, printer or saved as a digital image. Further, it was desired that the pictures created always remain within the picture area (i.e., never go off screen).

```
OPTIONS ASSIGN DEFINE DISPLAY EVAL PRINT
OUTPUT PRINTER INIT INPUT POP TERM DBG
OPERATORS + - / * ** NEG
DEFINED VARIABLES  A B C
EXPRESSION =
A/((((A+B)**(A+B))**((A+B)**(A+B)))*B)
```

$$\frac{A}{\left[[A+B]^{A+B}\right]^{[A+B]^{A+B}} * B}$$

```
 OVERFLOW
SELECT OPTION OR OPERATOR
```

FIG. 5.1--2-D math example--scope output.

```
OPTIONS ASSIGN DEFINE DISPLAY EVAL PRINT
OUTPUT PRINTER INIT INPUT POP TERM DBG
OPERATORS + - / * ** NEG
DEFINED VARIABLES  A B C D
EXPRESSION =  ((A+B)**(A+C))/(D/(B**(C+D)))
```

$$\frac{[A+B]^{A+C}}{\left[\frac{D}{B^{C+D}}\right]}$$

```
 6.150912E+07
SELECT OPTION OR OPERATOR
```

1929A10

FIG. 5.2--2-D math example--scope output.

```
((A+B)**(A+C))/(D/(B**(C+D)))


            A + C
--       --
| A + B |
--       --
---------------
  --         --
  |     D     |
  | -------   |
  |   C + D   |
  | B         |
  --         --
```

1929A18

FIG. 5.3--2-D math example--printer output.

```
A/((((A+B)**(A+B))**((A+B)**(A+B)))*B)

                        A
-----------------------------------------
                                  A + B
                         --       --
                         | A + B |
--                     -- --       --
|              A + B |
| --       --         |
| | A + B |           |                  * B
| --       --         |
--                     --                1929A17
```

FIG. 5.4--2-D math example--printer output.

THIS IS A TEST OF THE VERSION WITH EQUATIONS

This test illustrates the inclusion of equations in the input in linear form being converted to a two-dimensional form.

For example the equation ((A+B)**(A+C))/(D/(B**(C+D))) appears as:

```
            A + C
--      --
| A + B |
--      --
---------------
  --        --
  |     D    |
  | -------  |
  |   C + D  |
  | B        |
  --        --
```

and the equation A/((((A+B)**(A+B))**((A+B)**(A+B)))*B) appears as:

```
                      A
-------------------------------------
                                A + B
                        --      --
                        | A + B |
--                   -- --      --
|            A + B |
| --      --        |
| | A + B |         |                  * B
| --      --        |
--                   --
```

This was accomplished by modifying only the input procedure of the printing program; the modification simply calls the two-dimensional system implemented via GEMS. The output of the two-dimensional system is then used as input to the printing program. Clearly, any system implemented via GEMS could be similarly connected to the printing program; e.g. a drawing program for constructing figures, a flowcharting system to prepare flowcharts, etc.

1929A16

FIG. 5.5--Text printing with 2-D math.

```
  show systems
WYLBUR
JEGCG527
? jegcg527
WHAT'S THE MAGIC WORD? gems
OPTIONS ASSIGN DEFINE DISPLAY EVAL PRINT OUTPUT PRINTER
INIT INPUT POP TERM DBG
OPERATORS + - / * ** NEG
SELECT OPTION OR OPERATOR
  define
ANDDEF
ENTER VARIABLE NAME
  a
ENTER VALUE OF VARIABLE
ENTER VALUE
  2
DEFINED VARIABLES  A
EXPRESSION =
SELECT OPTION OR OPERATOR
  define
ANDDEF
ENTER VARIABLE NAME
  b
ENTER VALUE OF VARIABLE
ENTER VALUE
  3
DEFINED VARIABLES  A B
EXPRESSION =
SELECT OPTION OR OPERATOR
  +
SELECT LEFT OPERAND
  a
SELECT RIGHT OPERAND
  b
EXPRESSION =  A+B
SELECT OPTION OR OPERATOR
  eval
 5.000000E+00
SELECT OPTION OR OPERATOR
  **
SELECT EXPRESSION
  expression
SELECT EXPONENT
  expression
EXPRESSION =  (A+B)**(A+B)
SELECT OPTION OR OPERATOR
  print
          A + B
--      --
| A + B |
--      --
SELECT OPTION OR OPERATOR                                  1929A15
```

FIG. 5.6--2-D math example--typewriter output.

FIG. 5.6--(cont'd.)

```
  **
SELECT EXPRESSION
  expression
SELECT EXPONENT
  expression
EXPRESSION =  ((A+B)**(A+B))**((A+B)**(A+B))
SELECT OPTION OR OPERATOR
  print
                                   A + B
                        --      --
                        | A + B |
--                    -- --      --
|             A + B |
| --      --        |
| | A + B |         |
| --      --        |
--                  --
SELECT OPTION OR OPERATOR
  eval
 OVERFLOW
SELECT OPTION OR OPERATOR
  term
JEGCG527 HAS DIED. YOU WILL BE RETURNED TO WYLBUR
?
```

1929A14

The types of picture drawings of interest are illustrated excellently by Bongard (Bongard 1970). Further, a subset of PDL (Shaw 1968a,b; Miller and Shaw 1967; George 1968, 1969a; George and Miller 1968) was chosen as the language due to its known power of construction; its definition is given here for completeness. A picture is defined by:

PICTURE	::=	PICTURE + SUB-PICTURE
	::=	PICTURE - SUB-PICTURE
	::=	PICTURE & SUB-PICTURE
	::=	PICTURE * SUB-PICTURE
	::=	SUB-PICTURE
SUB-PICTURE	::=	¬ ELEMENT
	::=	# ELEMENT
	::=	ELEMENT
ELEMENT	::=	PRIMITIVE
	::=	(PICTURE)

where, a primitive is defined as any object with two distinguished points, a head and a tail (denoted by h and t respectively).

In all cases

Tail (S1 {+, -,*,& } S2) = Tail (S1)

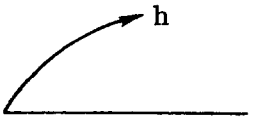
Head (S1 {+, -,*,& } S2) = Head (S2)

The binary concatenation operators specify how pictures are composed from more basic elements and are defined by:

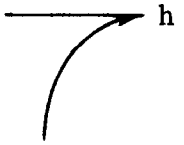
Let S1 be t ⟶ h and S2 be t ↗ h

then,

S1 + S2 = t ⟶ h (head to tail)

S1 & S2 = 

(tail to tail)

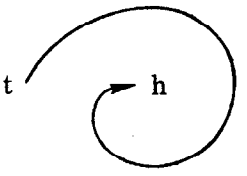
S1 - S2 = 

(head to head)

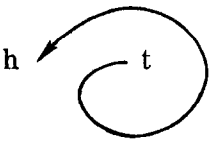
S1 * S2 = 

(tail to tail and head to head)

The unary operators are defined by:

Let S be 

then,

S = 

(tail/head reversal)

¬S = t t (blanking operator)

5.2.1 The Control Description

The functions to be defined are:

NAME	EXPLANATION
DISPLAY	Displays the string representation of a primitive or picture.
ASSIGN	Assigns the current expression to the T_S part of a primitive.

NAME	EXPLANATION
POP	Restores the current expression to the previous expression.
DEFINE	Allows primitives to be defined.
DRAW	Displays the graphical form of the current expression on the scope.
PRINT	Generates a digital representation (i.e., an array) of the graphical form of the current expression and then prints this array.
DBG	Complements the debug switch. More output is available in the debug mode.
TERM	Terminate the program.

The constructions to be defined are +, -, #, &, * and ¬ and were illustrated in Section 5.2. The external communication procedures are given in Appendix F and the control description is given in Appendix G.

5.2.2 The Definition Description

A primitive is any line drawing, group of points, arc of a circle or character string with two distinguished concatenation points (tail and head) and contained in a rectangular box whose size is specified. Thus, the attribute list contains the coordinates of the tail and head and the minimum and maximum excursions which define the rectangular box; these minimum and maximum are used to scale a picture for a given display area.

The construction features of the basic display file are line, point, alpha and arc. The definition description (Appendix G) specifies how a primitive is defined interactively and what the data types are required to be.

5.2.3 The Graphic Description

The graphic description consists of a syntax and an associated semantic description; for this application, only one semantic description is needed since any device can be driven from the resulting T_V.

The syntax description for the drawing language is given in Appendix G.

To perform the semantics for the concatenation operators, the translation feature of Section 5.1.3.3 is used. Thus, a sub-picture is translated to a particular position, the basic display files are merged, a new tail and head are calculated and new minimum and maximum excursions are calculated for each.

The output from the graphic evaluation (i.e., the parser and semantics) is a string of the form:

⟨SEP⟩ ⟨SEP⟩ TV

where TV = ⟨ATTR-LIST⟩ ⟨SEP⟩ ⟨BDF-FILE⟩

and ⟨ATTR-LIST⟩ contains the tail and head specifications and minimum and maximum x and y direction excursions. The constructions of the ⟨BDF-FILE⟩ are:

LINE { ⟨INTENSITY⟩ ⟨X⟩ ⟨Y⟩ } †

ALPHA ⟨NAME⟩

POINT { ⟨INTENSITY⟩ ⟨X⟩ ⟨Y⟩ }

ARC ⟨INTENSITY⟩ ⟨CENTER-X⟩ ⟨CENTER-Y⟩ ⟨RADIUS⟩ ⟨INITIAL ANGLE⟩ ⟨FINAL-ANGLE⟩

TRANS ⟨X⟩ ⟨Y⟩

UNTRAN ⟨X⟩ ⟨Y⟩

† { } means occurs zero or more times

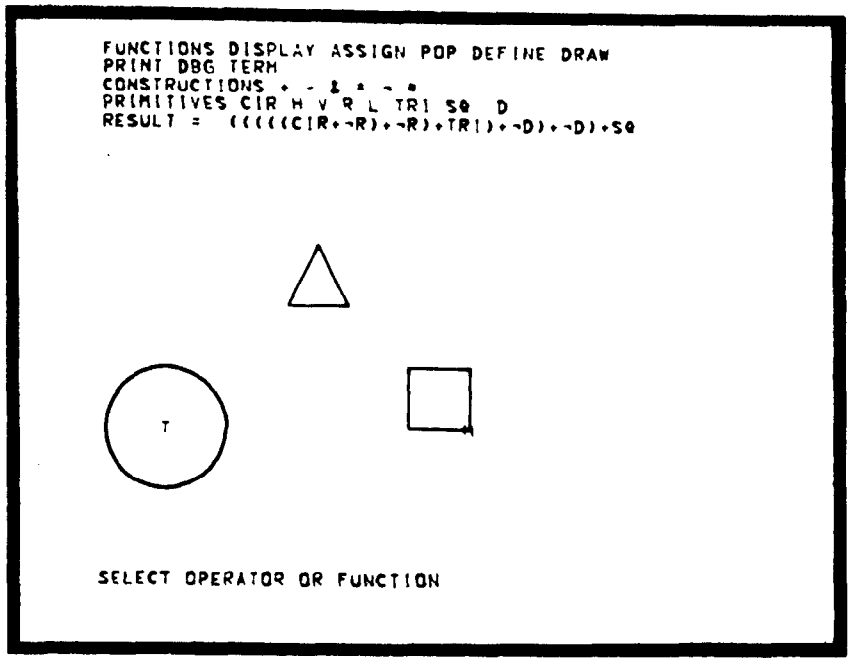
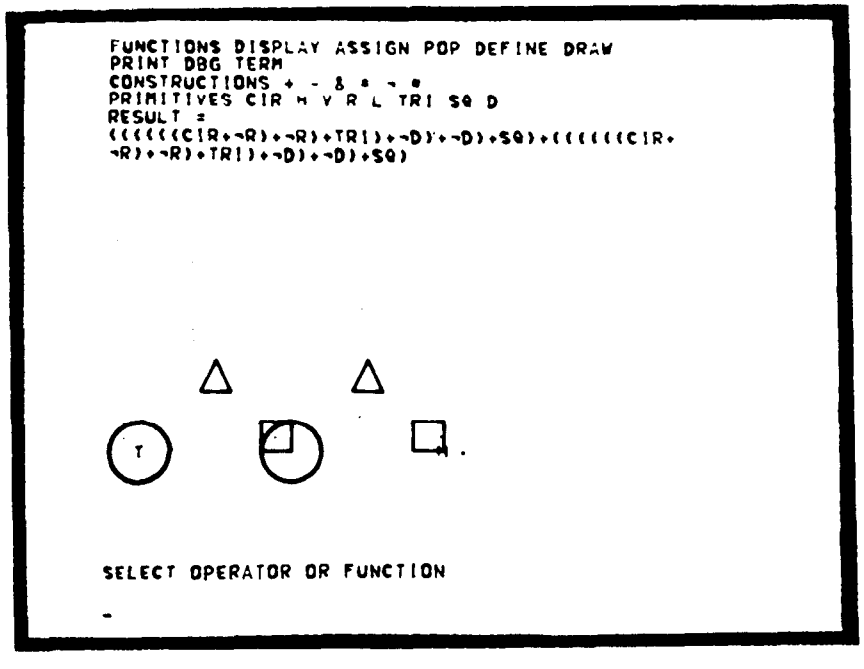


FIG. 5.7--Sample scope display from the drawing system.



1929A9

FIG. 5.8--Sample scope display from the drawing system.

```
(CIR+¯R)+TRI
```

```
      X
     XX
     X X
    X  X
    X   X
   X    X
   X     X
  X      X
  X       X
X         X
X          X
HXXXXXXXXXXX
```

```
          X
      XXXX XXXX
    XX         XX
   X             X
  XX             XX
 X                 X
 X                 X
X                   X
X                   X
X                   X
X                   X
X          T         X
X                   X
X                   X
X                   X
X                   X
 X                 X
 X                 X
  XX             XX
   X             X
    XX         XX
      XXXX XXXX
          X
```

1929A13

FIG. 5.9--Sample printer output from the drawing program.

```
(((CIR+¬R)+TRI)+¬D)+SQ
```

```
   X
  XX
  X X
 X  X
 X   X
X    X
X     X
X      X
X       X
XXXXXXXXXX
```

```
      X
    XXX XXX
  XX       XX
 XX         XX
XX           XX
X             X
X               X
X               X
X               X
X       T        X
X               X
X               X
X               X
X             X
XX           XX
XX         XX
XX       XX
    XXX XXX
      X
```

```
XXXXXXXXXX
X        X
X        X
X        X
X        X
X        X
X        X
X        X
X        X
XXXXXXXXXH
```

1929A12

FIG. 5.10--Sample printer output from the drawing program.

```
((((CIR+¬R)+TRI)+¬D)+SQ)+((((CIR+¬R)+TRI)+¬D)+SQ)
```

```
  X                          X
 XX                         XX
 X X                        X X
X  X                       X  X
X   X                      X   X
XXXXXX                     XXXXXX

   XXX                     XXXXXXX                    XXXXXX
  XXX XXX                  X XXXXXXX                  X    X
 XX     XX                 XXX  X  XX                 X    X
 X       X                 XX   X   X                 X    X
XX       XX                XX   X   XX                X    X
X    T    X                XXXXXX    X                XXXXXH
XX       XX                XX       XX
 X       X                 X        X
 XX     XX                 XX     XX
  XXX XXX                   XXX XXX
   XXX                        XXX                     1929A11
```

FIG. 5.11--Sample printer output from the drawing program.

The device drivers must accept these data construction types and are given in Appendix G.

5.2.4 Discussion of the Drawing System

Driver programs for the IBM 2250 Graphic Display and for the printer were written for this graphic language. The graphic display output is illustrated in Figures 5.7 and 5.8; the comparison between these two figures shows the effect of automatic scaling to avoid part of the drawing being off screen. The scaling used in linear; the same scale factors for both the x and y directions.

The printer driver forms a digital image of the entire picture before printing the array; the background and plotting characters are variable as is the digital image size. Sample displays on the printer are given in Figures 5.9 thru 5.11. Because the printer intercharacter and interline spacing are not equal, the circles become elipses and the squares become rectangles. A non-linear scale transformation would correct this and is allowable in the drawing system. Also notice the unnecessary points of the circle which illustrate computational problems with projection onto a coarse grid.

CHAPTER 6

CONCLUDING REMARKS

The GEMS model is a powerful and flexible tool for experimentation with graphical languages with a formal linguistic base. The control element is useful in many interactive applications. Further, the original goals of device independence and usability in an interactive and batch environment were illustrated in the examples. The method of implementation of GEMS (via SIMPLE) is modular and easy to modify, thus allowing GEMS to be easily changed, modified or extended.

6.1 Future Work

A. Proposed Studies

It has been suggested that a string description of a picture is an efficient way of storing or communicating the picture (Inselberg 1968; Shaw 1968a). This is also related to the subdivision of labor between a remote graphic terminal and the central computer (Morpurgo and Sami 1970). Another related question is how a microprogrammable graphic device can best be utilized; variable character sets for graphic devices have already been designed using a system which could be defined via GEMS (Rasmussen 1968). All of these questions could be phrased as a study of the transformation from a T_S to a T_V and from a T_V to a display file for a particular device; both of these transformations are accessible and modifiable within the GEMS model.

The model has always been illustrated with two-dimensional examples and it has been assumed to easily extend to three-dimensions (others have also

assumed this e.g., Shaw 1968a, 1969b and Inselberg 1968). The question of whether or not it is easily extendible to other dimensional systems still needs to be answered.

Baecker (1969a and b) has illustrated the use of motion in generation. A study to extend the model for motion in both generation and recognition is a challenging task; if motion can be included in any recognition model, significantly different kinds of pattern recognition problems can be attacked.

The symmetry between recognition and generation in the GEMS model was originally intended to allow synthesis testing, which has been suggested as a desirable feature (Kirsch 1968; Kulsrud 1968; Shaw 1968a and 1969b). This symmetry has not been adequately studied or utilized for a practical application.

B. Possible Applications

Some of the applications of GEMS which appear interesting to me are conformal mapping, pole-zero to root-locus transformations, circuit analysis, constraint systems like next to... (Inselberg 1968; Ledley and Wilson 1964), drawing and plotting languages (Breeding 1965; Frank 1968; Moorer 1970; Notley 1970; Sargent 1970; Schwinn 1967; van Dam 1967) and movie languages (Baecker 1969a and b; Citron and Whitney 1968; Knowlton 1964; Weiner and Anderson 1968).

In particular, I am initiating work to study the addition of graphics to APL. GEMS is particularly well suited for this, since it is designed to include operator languages easily and APL is such a language.

Thus, the work reported herein has been useful in areas other than graphics and is currently in active use to study graphics in another programming language.

BIBLIOGRAPHY

1. Ahuja, D.V. (1968). An Algorithm for Generating Spline-Like Curves. IBM Systems Journal, 7, 3 and 4, 206-217.

2. Ahuja, D.V. and Coons, S.A. (1968). Geometry for Construction and Display. IBM Systems Journal, 7, 3 and 4, 188-205.

3. Appel, A. (1968). Modeling in Three Dimensions. IBM Systems Journal, 7, 3 and 4, 310-321.

4. Appel, A., Dankowski, T.P. and Dougherty, R.L. (1968). Aspects of Display Technology. IBM Systems Journal, 7, 3 and 4, 176-187.

5. Anderson, Robert H. (1968). Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics. The Division of Engineering and Applied Physics, Harvard University. Also in Interactive Systems for Experimental Applied Mathematics , Melvin Klerer and Juris Reinfelds (Eds.), 436-459.

6. Anderson, R., Bator, R., Gagan, R., Meads, J. and Metrick, L. (1970). Interactive Specification of Data Displays. Wolf Research and Development Corp. for Goddard Space Flight Center, NASA-CR-1628.

7. Arthurs, E., Bartlett, W.S., Ladd, D.J., Salmon, R.L. and Whipple, J.H. (1970). Picturelab - An Interactive Facility for Experimentation in Picture Processing. Proceedings of the 1970 Spring Joint Computer Conference , 36,267-273.

8. Baecker, Ronald Michael (1969a). Interactive Computer-Mediated Animation. Project MAC, Massachusetts Institute of Technology, MAC-TR-61.

9. Baecker, Ronald M. (1969b). Picture-Driven Animation. Proceedings of the 1969 Spring Joint Computer Conference , 34, 273-288.

10. Balzer, R.M. (1967). Dataless Programming. The RAND Corporation, Memorandum RM-5290-ARPA. Condensed version appeared in Proceedings of the 1967 Fall Joint Computer Conference , 31, 535-544.

11. Barlett, W.S., Busch, K.J., Flynn, M.L. and Salmon, R.L. (1968). SIGHT, A Satellite Interactive Graphic Terminal. Proceedings of the 23rd ACM National Conference , P-68, 499-509.

12. Baskin, Herbert B. (1970). Interactive Graphics and Computer Aided Design Techniques. Proceedings of the Society of Photographic Scientists and Engineers Seminar on Computer Handling of Graphical Information , 121-128.

13. Baskin, H.B. and Morse, S.P. (1968). A Multilevel Modeling Structure for Interactive Graphic Design. IBM Systems Journal, 7, 3 and 4, 218-228.

14. Bernstein, M.I. and Williams, T.G. (1968). A Two-Dimensional Programming System. Proceedings of IFIP Congress 1968 ,586-592.

15. Blackwell, Frederick W. and Anderson, Robert H. (1969). An On-Line Symbolic Mathematics System Using Hand-Printed Two-Dimensional Notation. Proceedings of the 24th ACM National Conference , P-69, 551-557.

16. Boehm, B.W., Lamb, V.R., Mobley, R.L. and Rieber, J. E. (1969). POGO: Programmer-Oriented Graphics Operation. The RAND Corporation, Memorandum RM-5825-PR. Also in Proceedings of the 1969 Spring Joint Computer Conference , 34, 321-330.

17. Bongard, N. (1970). Pattern Recognition (Ed. J.K. Hawkins). Spartan Books.

18. Bowman, Sally and Linkhalter, Richard A. (1968). Graphical Data Management in a Time-Shared Enviornment. Proceedings of the 1968 Spring Joint Computer Conference , 32, 353-362.

19. Breeding, Kenneth James (1965). Grammar for a Pattern Description Language. Department of Computer Science, University of Illinois, Report No. 177.

20. Breme, Hans J. (1970). Contour Map and Terrain Intervisibility Processing: A Literature Survey. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 403-11.

21. Brown, G.D., Bush, C.H. and Berman, R.A. (1968). The Integrated Graphics System for the S-C 4060: I. User's Manual. The RAND Corporation, Memorandum RM-5660-PR, December, 1968.

22. Brown, R.M., Fisherkeller, Mary Anne, Gromme, A.E. and Levy, J.V. (1966). The SLAC High-Energy Spectrometer Data Acquisition and Analysis System. Proceedings of the IEEE, 54, 12(December), 1730-1734.

23. Butt, Edgar B. and Snively, James W. (1969). The PAX II Picture Processing System. Computer Science Center, University of Maryland, September 1969.

24. Calvert, Thomas W. (1968). Projections of Multidimensional Data for Use in Man-Computer Graphics. Proceedings of the 1968 Fall Joint Computer Conference , 33, 227-231.

25. Carlbom, Ingrid (1968). Algorithms for Transforming PDL-Expressions into Standard Form and into a Primitive Connection Matrix. Stanford Linear Accelerator Center Computation Group, CGTM 38.

26. Chen, F.C. and Dougherty (1968). A System for Implementing Interactive Applications. IBM Systems Journal, 7, 3 and 4, 257-270.

27. Childs, David L. (1968). Description of a Set-Theoretic Data Structure. Proceedings of the 1968 Fall Joint Computer Conference , 33, 557-564.

28. Christensen, Carl and Pinson, Elliot N. (1967). Multi-Function Graphics for a Large Computer System. Proceedings of the 1967 Fall Joint Computer Conference , 31, 697-711.

29. Citron, J. and Whitney, John H. (1968). CAMP-Computer Assisted Movie Production. Proceedings of the 1968 Fall Joint Computer Conference , 33, 1299-1305.

30. Clowes, Dr. M.B. (1967a). Perception, Picture Processing and Computers. Machine Intelligence I , Collins, N.L. and Michie, D. (Ed), Oliver & Boyd, London, 181-197.

31. Clowes, M.B. (1967b). A Generative Picture Grammar. Computing Research Section, Commonwealth Scientific and Industrial Research Organization, Seminar Paper No. 6.

32. Clowes, M.B. (1969). Pictorial Relationships- a Syntactic Approach. Machine Intelligence 4 , Americal Elsevier, 361-383.

33. Cohen, Doron J. and Gotlieb, C.C. (1970). A List Structure Form of Grammars for Syntactic Analysis. Computing Surveys, 2, 1(March), 65-82.

34. Comba, P.G. (1968). A Language for Three-Dimensional Geometry. IBM Systems Journal, 7, 3 and 4, 292-307.

35. Cotton, Ira W. and Greatorex, Frank S., Jr. (1968). Data Structures and Techniques for Remote Computer Graphics. Proceedings of the 1968 Fall Joint Computer Conference , 33, 533-544.

36. Csuri, Charles and Shaffer, James (1968). Art, Computers and Mathematics. Proceedings of the 1968 Fall Joint Computer Conference , 33, 1293-1298.

37. Dahl, Ole-Johan, Myhrhaug, Bjorn and Nygaard, Kristen (1968). SIMULA 67 Common Base Language. Norwegian Computing Center.

38. Davis, M.R. and Ellis, T.O. (1964). The RAND Tablet: A Man-Machine Graphical Communication Device. The RAND Corporation, Memorandum RM-4122-ARPA.

39. Dertouzos, Michael L. (1967). PHASEPLOT: An On-Line Graphical Display Technique. IEEE Transactions on Electronic Computers, EC-16, 2(April), 203-209.

40. Duda, Richard O. and Hart, Peter E. (1968). Experiments in the Recognition of Hand-Printed Text: Part II--Context Analysis. Proceedings of the 1968 Fall Joint Computer Conference ,33,1139-1149.

41. Duffin, John David (1970). A Language for Line Drawing. Department of Computer Science, University of Toronto.

42. Eden, Murray (1961). On the Formalization of Handwriting. Proceedings of Symposia in Applied Mathematics , American Mathematical Society, 12, 83-88.

43. Eden, Murray (1962). Handwriting and Pattern Recognition. IRE Transactions on Information Theory, IT-8, 2(February), 160-166.

44. Ellis, T.O. and Sibley, W.L. (1966). On the Development of Equitable Graphic I/O. The RAND Corporation, P-3415.

45. Ellis, T.O. and Sibley, W.L. (1967). On the Problem of Directness in Computer Graphics. Proceedings from Emerging Concepts in Computer Graphics .

46. Engelman, C. (1965). MATHLAB: A Program for On-Line Assistance in Symbolic Computations. Proceedings of the 1965 Fall Joint Computer Conference , 27,413-422.

47. Engelman, C. (1968). MATHLAB 68. Proceedings of IFIP Congress 1968 , 462-467.

48. Evans, Thomas G. (1969). Descriptive Pattern-Analysis Techniques: Potentialities and Problems. Methodologies of Pattern Recognition (Ed. Satosi Watanabe) , Academic Press.

49. Feder, Jerome (1966a). The Linguistic Approach to Pattern Analysis: A Literature Survey. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 400-133.

50. Feder, Jerome (1966b). Linguistic Specification and Analysis of Classes of Patterns. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 400-147.

51. Feder, Jerome (1967). Languages, Automata and Classes of Chain-Encoded Patterns. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 400-165.

52. Feder, Jerome (1969). Linguistic Specification and Analysis of Classes of Line Patterns. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 403-2, April, 1969.

53. Feldman, Jerome A. (1964). A Formal Semantics for Computer-Oriented Languages. Carnegie Institute of Technology.

54. Feldman, Jerome A. (1966). A Formal Semantics for Computer Languages and its Application in a Compiler-Compiler. Comm. ACM, 9, 1(January), 3-9.

55. Feldman, Jerome A. and Gries, David (1967). Translator Writing Systems. Computer Science Department, Stanford University, Technical Report No. CS 69. Also appeared in Comm. ACM, 11(1968), 2(February), 77-113.

56. Fischer, Michael J. (1969). Some Properties of Precedence Languages. ACM Symposium on Theory of Computing, 181-190.

57. Fisher, David A. (1970). Control Structures for Programming Languages. Department of Computer Science, Carnegie-Mellon University.

58. Frank, Amalie J. (1968). B-LINE, Bell Line Drawing Language. Proceedings of the 1968 Fall Joint Computer Conference , 33, 179-191.

59. Freeman, Herbert (1961). On the Encoding of Arbitrary Geometric Configurations. IRE Transactions on Electronic Computers, EC-10, 2(June), 260-268.

60. Fridh, R. Conny W. (1967). An Attempt to Put Parts of Roberts' "Machine Perception of Three-Dimensionsl Solids" into the Framework of Picture Calculus. Stanford Linear Accelerator Center Computation Group, CGTM 28.

61. Gear, C.W. (1969). An Interactive Graphic Modeling System. Department of Computer Science, University of Illinois, Report No. 318.

62. George, J.E. (1967a). SARPSIS: Syntax Analyzer, Recognizer, Parser and Semantic Interpretation System. Stanford Linear Accelerator Center, CGTM 34, November 15, 1967.

63. George, J.E. (1967b). The SPIRES Scope Demonstration System. Stanford Linear Accelerator Center, CGTM 33, November 15, 1967.

64. George, James E. (1967c). Picture Generation Using the Picture Calculus. Stanford Linear Accelerator Center Computation Group, GSG 50.

65. George, James E. (1968). CALGEN-An Interactive Picture Calculus Generation System. Computer Science Department, Stanford University, Technical Report No. 114.

66. George, James E. (1969a). The System Specification of GLAF: A Linear String Graphical Language Facility. Stanford Linear Accelerator Center, GSG-61, February, 1969.

67. George, J.E. (1969b). GEMS: A Graphic Experimental Meta-System. Stanford Linear Accelerator Center, GSG 63, June, 1969.

68. George, J.E. (1969c). Rules for Transforming a Grammar to a Simple Precedence Grammar Utilizing Artificial Productions. Stanford Linear Accelerator Center Computation Group, GSG-62, July, 1969.

69. George, J.E. (1971a). PRINT--A Text Formatting Program. Stanford Linear Accelerator Certer Computation Group, CGTM-128.

70. George, James E. (1971b). SIMPLE--A Simple Precedence Translator Writing System. Stanford Linear Accelerator Center, SLAC-133. Also, Computer Science Department, Stanford University, Technical Report to be issued.

71. George, J.E. and Miller, W.F. (1968). String Descriptions of Data for Display. 9th National Symposium on Information Display , 143-147.

72. George, James E. and Saal, Harry J. (1971). A Command Language Meta-System. Fourth Hawaii International Conference on System Sciences , 483-485. Also Stanford Linear Accelerator Center, SLAC-PUB-844.

73. Glucksman, Herbert A. (1967). Classification of Mixed-Font Alphabetics by Characteristic Loci. Digest of the First Annual IEEE Computer Conference , 16C51, 138-141.

74. Gray, James (1969). Precedence Parsers for Programming Languages. Department of Computer Science, University of California.

75. Gray, J.C. (1967). Compound Data Structure for Computer Aided Design; A Survey. Proceedings of the 1967 ACM National Meeting , P-67, 355-365.

76. Green, R. Elliot (1970). Computer Graphics. Computer Aided Design Spring Supplement, 29-60.

77. Grenander, Ulf (1969). Foundations of Pattern Analysis. Quarterly of Applied Mathematics, 27, 1(April), 1-56.

78. Gries, David (1967). The Use of Transition Matrices in Compiling. Computer Science Department, Stanford University, Technical Report No. CS 57. Also appeared in Comm. ACM, 11(1968), 1(January), 26-34.

79. Groner, Gabriel F. (1966). Real-Time Recognition of Handprinted Text. The RAND Corporation, Memorandum RM-5016-ARPA.

80. Groner, G.F. (1968). Real-Time Recognition of Handprinted Text: Program Documentation. The RAND Corporation, Memorandum RM-5550-ARPA.

81. Guzman-Arenas, Adolfo (1967). Some Aspects of Pattern Recognition by Computer. Project MAC, Massachusetts Institute of Technology, MAC-TR-37.

82. Guzman, Adolfo (1969). Decomposition of a Visual Scene into Three Dimensional Bodies. Massachusetts Institute of Technology, Project MAC, MAC-M-391.

83. Hall, David J. and Wensley, John H. (1968). META- A Meta-Compiler system and a High-Level Language for Man/Machine Display Programs. Stanford Research Institute.

84. Hall, D.J., Ball, G.H., Wolf, D.E. and Eusebis, J.W. (1968). Promenade-An Interactive Graphics Pattern-Recognition System. Information Display, November/December, 1968, 45-49. Stanford Research Institute, SRI Project 6737.

85. Henderson, Dugald Austin, Jr. (1967). A Graphics Display Language. Department of Computer Science, University of Illinois, Report No. 240.

86. Hodes, Lou (1961). Machine Processing of Line Drawings. Lincoln Laboratory, Massachusetts Institute of Technology, Technical Report No. 83.

87. Hodes, Louis (1970). A Programming System for the On-Line Analysis of Biomedical Images. Comm. ACM, 13, 5(May), 279-283.

88. Huggins, W.H. and Entwisle, Doris R. (1969). Computer Animation for the Academic Community. Proceedings of the 1969 Spring Joint Computer Conference ,34,623-627.

89. Hurwitz, A., Citron, J.P. and Yeaton, J.B. (1967). GRAF: Graphic Additions to FORTRAN. Proceedings of the 1967 Spring Joint Computer Conference , 30, 553-557.

90. Inselberg, A.D. (1968). SAP: A Model for the Syntactic Analysis of Pictures. Sever Institute of Technology, Washington University, St. Louis, Missouri.

91. Johnson, C.I. (1968). Principles of Interactive Systems. IBM Systems Journal, 7, 3 and 4, 147-173.

92. Kilgour, A.C. (1967). Computer-Aided Design Project. University of Edinburgh, Memo CAD-R-10.

93. Kirsch, Russell A. (1964). Computer Interpretation of English Text and Picture Patterns. IEEE Transactions on Electronic Computers, EC-13, 4(August), 363-376.

94. Kirsch, Russell A. (1968). Picture Syntax. Pattern Recognition Laveen N. Kanal (ed.), 183-185.

95. Klerer, Melvin and May, Jack (1965). Two-Dimensional Programming. Proceedings of the 1965 Fall Joint Computer Conference , 27, 63-75.

96. Klerer, Melvin and Grossman, Fred (1967). Further Advances in Two-Dimensional Input-Output by Typewriter Terminals. Proceedings of the 1967 Fall Joint Computer Conference , 31, 675-687.

97. Knoke, Peter J. and Wiley, Richard G. (1967). A Linguistic Approach to Mechanical Pattern Recognition. Digest of the First Annual IEEE Computer Conference , 16C51, 142-144.

98. Knowlton, K.C. (1964). A Computer Technique for Producing Animated Movies. Proceedings of the 1964 Fall Joint Computer Conference , 25,67-87.

99. Kopel, P.S. (1968). Interactive Computer Graphics. Bell Laboratories Record, 46, 6(June), 189-196.

100. Krakauer, Lawrence Jay (1964). Syntax and Display of Printed Format Mathematical Formulas. Massachusetts Institute of Technology.

101. Kulsrud, H.E. (1968). A General Purpose Graphic Language. Comm. ACM, 11, 4(April), 247-254.

102. Lang, C.A., Polansky, R.B. and Ross, D.T. (1965). Some Experiments with an Algorithmic Language. Electronic Systems Laboratory, Department of Electrical Engineering, Massachusetts Institute of Technology, ESL-TM-220.

103. Learner, A. and Lim, A.L. (1970). A Note on Transforming Context-Free Grammars to Wirth-Weber Precedence Form. The Computer Journal, 13, 2(May), 142-144.

104. Ledley, Robert S. and Wilson, James B. (1964). Concept Analysis by Syntax Processing. Proceedings of the American Documentation Institute Annual Meeting , 1, 1-8.

105. Ledley, R.S., Rotolo, L.S., Golab, T.J., Jacobsen, J. D., Ginsberg, M.D. and Wilson, J.B. (1965). FIDAC: Film Input to Digital Automatic Computer and Associated Syntax-Directed Pattern-Recognition Programming System. Optical and Electro-Optical Information Processing , Tippet et. al. (Ed), M.I.T. Press, Cambridge, Massachasetts, 591-613.

106. Ledley, R.S., Jacobsen, J. and Belson, M. (1966). BUGSYS: A Programming System for Picture Processing--Not for Debugging. Comm. ACM, 9, 2(February), 79-84.

107. Leinius, Ronald Paul (1970). Error Detection and Recovery for Syntax Directed Compiler Systems. University of Wisconsin.

108. Licklider, J.C.R. (1969). A Picture is Worth a Thousand Words-and It Costs..... Proceedings of the 1969 Spring Joint Computer Conference , 34, 617-621.

109. Lipkin, Lewis E., Watt, William C. and Kirsch, Russell A. (1966). The Analysis, Synthesis and Description of Biological Images. Annals of the New York Academy of Sciences , 128, 3(January), 984-1012.

110. Martin, William A. (1967). Symbolic Mathematical Laboratory. Massachusetts Institure of Technology, Project MAC, MAC-TR-36.

111. Mathews, M.V. and Miller, Joan E. (1965). Computer Editing, Typesetting and Image Generation. Proceedings of the 1965 Fall Joint Computer Conference , 27, 389-398.

112. Mezei, L. (1968). Sparta, A Procedure Oriented Programming Language for the Manipulation of Arbitrary Line Drawings. Proceedings of the 1968 IFIP Congress .

113. Millen, Jonathan K. (1968). CHARYBDIS: A LISP Program to Display Mathematical Expressions on Typewriter-Like Devices. Interactive Systems for Experimental Applied Mathematics , Melvin Klerer and Juris Reinfelds (Eds.), 155-163.

114. Miller, W.F. (1969). Hardware and Software for Effective Man-Machine Interaction in the Laboratory. Stanford Linear Accelerator Center, SLAC-PUB-573.

115. Miller, W.F. and Shaw, Alan C. (1967). A Picture Calculus. Stanford Linear Accelerator Center, Stanford University, SLAC-PUB-358. Also in Proceedings of Emerging Concepts in Computer Graphics .

116. Miller, W.F. and Shaw, A.C. (1968). Linguistic Methods in Picture Processing-A Survey. Proceedings of the 1968 Fall Joint Computer Conference , 33, 279-290.

117. Moorer, Andy (1970). DPY - A Device Independent Graphics Package. Stanford Artificial Intelligence Laboratory, Stanford University, Operating Note Number 63.

118. Morpurgo, R. and Sami M. (1970). Some Solutions to the Problem of Defining and Compiling Graphic Languages. Revue Francaise d' Informatique et de Recherche Operationnelle, Numero B-1, 21-30.

119. Morrison, Ronald A. (1967). Graphic Language Translation with a Language Independent Processor. Proceedings of the 1967 Fall Joint Computer Conference , 31, 723-731.

120. Munson, John H. (1968). Experiments in the Recognition of Hand-Printed Text: Part I--Character Recognition. Proceedings of the 1968 Fall Joint Computer Conference ,33,1125-1138.

121. McAfee, J. and Presser, L. (1970). An Algorithm for the Design of Simple Precedence Grammars. Department of Electrical Engineering, University of California at Santa Barbara.

122. McKeeman, W.M. (1966). An Approach to Computer Language Design. Computer Science Department, Stanford University, Technical Report No. CS 48.

123. Narasimhan, R. (1962). A Linguistic Approach to Pattern Recognition. Digital Computer Laboratory, University of Illinois, Report No. 121.

124. Narasimhan, R. (1964). Labeling Schemata and Syntactic Descriptions of Pictures. Information and Control, 7, 151-179.

125. Narasimhan, R. (1966a). Syntax-Directed Interpretation of Classes of Pictures. Comm. ACM 9, 3(March), 166-173.

126. Narasimhan, R. and Reddy, V.S.N. (1966b). A Generative Model for Hand-Printed English Letters and its Computer Implementation. Computer Group, Research and Development, Tata Institute of Fundamental Research, Colaba, Bombay, Techinical Report No. 12.

127. Newman, William M. (1968). A System for Interactive Graphical Programming. Proceedings of the 1968 Spring Joint Computer Conference , 32, 47-54.

128. Ninke, William H. (1965). GRAPHIC 1 - A Remote Graphical Display Console System. Proceedings of the 1965 Fall Joint Computer Conference , 17, 839-846.

129. Ninke, W.H. (1968). The Growth of Computer Graphics at Bell Laboratories. Bell Laboratories Record, 46, 6(June), 180-188.

130. Ninke, William H. (1969). Interactive Computer Graphics: Some Interpolations and Extrapolations. Pertinent Concepts in Computer Graphics , Proceedings of the Second University of Illinois Conference on Computer Graphics (Ed. M. Faiman and J. Nievergelt), University of Illinois Press, 429-439.

131. Noll, A. Michael (1968). Computer Animation and the Fourth Dimension. Proceedings of the 1968 Fall Joint Computer Conference ,33, 1279-1283.

132. Notley, M.G. (1970). A Graphical Picture Drawing Language. The Computer Bulletin, 14, 3(March), 68-74.

133. Pankhurst, R.J. (1968). GULP-A Compiler-Compiler for Verbal and Graphic Languages. Joint Computer Aided Design Project, University of Cambridge, Number 68-274. Also appeared in Proceedings of the 23rd ACM National Conference , P-68, 405-421.

134. Parker, Edwin B. (1967). SPIRES 1967 Annual Report. Institute for Communication Research, Stanford University, December, 1967.

135. Pfaltz, John L., Snively, James W. and Rosenfeld, Azriel (1968). Local and Global Picture Processing by Computer. Pictorial Pattern Recognition , Proceedings of the Symposium on Automatic Photointerpretation, Cheng, George C., Ledley, Robert S., Pollack, Donald K. and Rosenfeld, Azriel (Ed).

136. Presser, L. (1968). The Structure, Specifications and Evaluation of Translators and Translator Writing Systems. Department of Engineering, University of California at Los Angeles, Report No. 68-51.

137. Presser, L. and Melkanoff M.A. (1969). Transformation to Simple-Precedence. Second Hawaii International Conference on System Science , 695-698.

138. Rasmussen, Harold S. (1968). Graphic-Aided Design of a Graphic Device. 9th National Symposium on Information Display , 61-68.

139. Richardson, Fontaine K. (1968). Graphical Specification of Computation. Department of Computer Science, University of Illinois, Report No. 257.

140. Rippy, D.E. and Humphries, D.E. (1965). Magic-A Machine for Automatic Graphics Interface to a Computer. Proceedings of The 1965 Fall Joint Computer Conference , 27, 819-830.

141. Roberts, L.G. (1965a). Graphical Communication and Control Languages. Proceedings of the 2nd Congress on the Information System Sciences , 211-217.

142. Roberts, L.G. (1965b). Machine Perception of Three-Dimensional Solids. Optical and Electro-Optical Information Processing , Tippet et. al. (Ed), M.I.T. Press, Cambridge, Massachusetts, 159-197.

143. Roberts, Lawrence G. (1966). A Graphical Service System with Variable Syntax. Comm. ACM, 9, 3(March), 173-176.

144. Rosenfeld, Azriel (1969a). Picture Processing by Computer , Academic Press, Chapt. 10(Picture Description and Picture Languages).

145. Rosenfeld, Azriel (1969b). Picture Processing by Computer. Computing Surveys 1,3(September), 147-174.

146. Ross, Douglass T. and Rodriguez, Jorge E. (1963). Theoretical Foundations for the Computer-Aided Design System. Proceedings of The 1963 Spring Joint Computer Conference , 23, 305-322.

147. Rully, A.D. (1968). A Subroutine Package for FORTRAN. IBM Systems Journal, 7, 3 and 4, 248-256.

148. Ruyle, Adrian (1968). The Development of Systems for On-Line Mathematics at Harvard. Interactive Systems for Experimental Applied Mathematics , Melvin Klerer and Juris Reinfelds (Eds.), 241-254.

149. Sandewall, Erik (1968). Use of Ambiguity Logic in the Picture Description Language. Stanford Linear Accelerator Center Computation Group, GSG 52.

150. Sargent Murray III (1970). SCROLL - A Pattern Recording Language. Proceedings of the 1970 Spring Joint Computer Conference , 36,525-537.

151. Schlumberger, Maurice and Wyeth, David (1971). A Multi-Editor System. Computer Science Department, Stanford University, CS 293 Report. Also, Stanford Linear Accelerator Center, CGTM 127.

152. Schwartz, Judah L. and Taylor, Edwin F. (1968). Computer Displays in the Teaching of Physics. Proceedings of the 1968 Fall Joint Computer Conference , 33, 1285-1292.

153. Schwinn, Peter M. (1967). A Problem Oriented Graphic Language. Proceedings of the 1967 ACM National Meeting , P-67, 471-477.

154. Sharma, Onkar P. (1970). A Syntactic Model for On-Line, Real-Time Description, Analysis and Generation of Handdrawn Patterns. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 403-12.

155. Sharma, Onkar P. (1970). A System for Syntax Specification in On-Line Pattern Recognition. Laboratory for Electroscience Research, Department of Electrical Engineering, New York University, Technical Report 403-13.

156. Shaw, Alan C. (1966). Lecture Notes on a Course in Systems Programming. Computer Science Department, Stanford University, Technical Report No. 52.

157. Shaw, Alan C. (1968a). The Formal Description and Parsing of Pictures. Computer Science Department, Stanford University, Technical Report No. CS 94.

158. Shaw, Alan C. (1968b). A Formal Picture Description as a Basis for Picture Processing Systems. Stanford Linear Accelerator Center, SLAC-PUB-402, appeared in Information and Control , 12,1(January 1969), 9-52.

159. Shaw, Alan C. (1969a). Parsing of Graph-Representable Pictures. Department of Computer Science, Cornell University, Technical Report 69-34, April, 1969.

160. Shaw, Alan C. (1969b). On the Interactive Generation and Interpretation of Artificial Pictures. Stanford Linear Accelerator Center, SLAC-PUB-664. Presented at the 1969 ACM/SIAM/IEEE Conference on Mathematical and Computer Aids to Design, October, 1969.

161. Sibley, E.H., Taylor, R.W. and Ash, W.L. (1970). The Case for a Generalized Graphic Problem Solver. Proceedings of the 1970 Spring Joint Computer Conference , 36, 11-17.

162. Sibley, Edgar H., Taylor, Robert W. and Gordon, David G. (1968). Graphical Systems Communication: An Associative Memory Approach. Proceedings of the 1968 Fall Joint Computer Conference , 33, 545-555.

163. Skinner, Frank D. (1966). Computer Graphics-Where Are We? Datamation, 12, 5(May), 28-31.

164. Smura, E.J. (1968). Graphical Data Processing. Proceedings of The 1968 Spring Joint Computer Conference , 32, 111-118.

165. Standish, T.A. (1967). A Data Definition Facility for Programming Languages. Carnegie Institute of Technology.

166. Sutherland, I.E. (1963). Sketchpad: A Man-Machine Graphical Communication System. Lincoln Laboratory, Massachusetts Institute of Technology, Technical Report No. 296. Condensed version appeared in Proceedings of the 1963 Spring Joint Computer Conference , 23, 329-346.

167. Sutherland, Ivan E. (1966). Computer Graphics: Ten Unsolved Problems. Datamation, 12, 5(May), 22-27.

168. Sutherland, Ivan E. (1970). Computer Displays. Scientific American, 222, 6(June), 57-81.

169. Sutherland, W.R. (1966). On-Line Graphical Specification of Computer Procedures. Lincoln Laboratory, Massachusetts Institute of Technology, Technical Report 405.

170. Sutherland, William R. (1967). Language Structure and Graphical Man-Machine Communication. Proceedings of the 3rd Congress on the Information Systems Science and Technology , 29-31.

171. Symonds, A.J. (1968). Auxillary-Storage Associative Data Structure for PL/1. IBM Systems Journal, 7, 3 and 4, 229-245.

172. Thomas, Eugene M. (1967). GRASP-A Graphic Service Program. Proceedings of the 1967 ACM National Meeting , P-67, 395-402.

173. Tichenor, D.A. (1970). String Analysis by Simple Precedence Parsers. Data Processing Division, Sandia Laboratories, Livermore, SCL-DR-70-237.

174. van Dam, Andries and Evans, David (1967). A Compact Data Structure for Storing, Retrieving and Manipulating Line Drawings. Proceedings of the 1967 Spring Joint Computer Conference , 30, 601-610.

175. Weiner, D.D. and Anderson, S.E. (1968). A Computer Animation Movie Language for Educational Motion Pictures. Proceedings of the 1968 Fall Joint Computer Conference , 33, 1317-1320.

176. Wells, Mark B. (1961). MADCAP: A Scientific Compiler for a Displayed Formula Textbook Language. Comm. ACM, 4, 1(January), 31-36.

177. Wells, Mark B. (1963). Recent Improvements in MADCAP. Comm. ACM, 6, 11(November), 674-678.

178. Winkless, Nels and Honore, Paul (1968). What Good Is A Baby? Proceedings of the 1968 Fall Joint Computer Conference , 33, 1307-1315.

179. Winston, Patrick H. (1968). Holes. Massachusetts Institute of Technology, Project MAC, AI Memo 163.

180. Wirth, Niklaus (1965). Find Precedence Functions. Comm. ACM, Algorithm 265, 8, 10(October), 604-605.

181. Wirth, Niklaus and Weber, Helmut (1966a). EULER: A Generalization of ALGOL, and its Formal Definition: Part I. Comm. ACM, 9, 1(January), 13-25.

182. Wirth, Niklaus and Weber, Helmut (1966b). EULER: A Generalization of ALGOL, and its Formal Definition: Part II. Comm. ACM, 9, 2(February), 89-99.

183. Yarbrough, Lynn D. (1968). A User's Guide to CAFE. ARCON Corporation, Wakefield, Massachusetts.

184. Zahn, C.T. (1969). A Formal Description for Two-Dimensional Patterns. Stanford Linear Accelerator Center, SLAC-PUB-538.

APPENDIX A -- CONTROL LANGUAGE SPECIFICATION

SYNTAX

```
//GO.SYNDATA DD *
 SYM(1)='CONTROLA' SYM(2)='FUNCTION-PART' SYM(3)='CONSTRUCTION-PART'
 ERRORSCAN='*END*' SEQUENCE='CON_LANG' PARSER_NAME='CON_LANG'
 QUOTES='''
 SEMANT_NAME='SEMANT' TERMINAL='*END-CONTROL*' MLIM=75 NLIM=75 MMLIM=75
/*
//GO.SYNTAX DD *
*SYNTAX*
CON_LANG *::=* CONTROLA *CODE* DESCRIPTION *;*
CONTROLA *::=* *CONTROL* STRINGA DATA UNARY *;*
STRINGA *::=* STRING *;*
UNARY *::=* GROUP *NO-SEMANT* *;*
GROUP *::=* WORD *;*
*::=* GROUP WORD *;*
DATA *::=* DATUM *NO-SEMANT* *;*
DATUM *::=* ITEM *;*
*::=* DATUM ITEM *;*
ITEM *::=* SEP *=* STRING *;*
 *::=* QUOTES *=* STRING *;*
 *::=* CONNECTOR *=* STRING *;*
 *::=* CSEP *=* STRING *;*
 *::=* LEFT_PAREN *=* STRING *;*
 *::=* RIGHT_PAREN *=* STRING *;*
 *::=* TERMINATOR *=* STRING *;*
 *::=* MARKER *=* STRING *;*
 *::=* ASEP *=* STRING *;*
 *::=* MAX_PRIM *=* INTEGER *;*
 *::=* LENGTH_PRIM *=* INTEGER *;*
DESCRIPTION *::=* FUNCTION-PART  *;*
 *::=* FUNCTION-PART CONSTRUCTION-PART  *;*
FUNCTION-PART *::=* FUNA OP-COMMANDS *END* *;*
FUNA *::=* *FUNCTION* STRING *;*
CONSTRUCTION-PART *::=* CONA OP-COMMANDS *END* *;*
CONA *::=* *CONSTRUCTION* STRING *;*
OP-COMMANDS *::=* OPCOM *NO-SEMANT* *;*
OPCOM *::=* OP-COMMAND *;*
 *::=* OPCOM OP-COMMAND *;*
OP-COMMAND *::=* OPCON OPERANDS *CODE* *NO-SEMANT* *;*
OPCON *::=* *NAME* *=* WORD *;*
OPERANDS *::=* OPERA *NO-SEMANT* *;*
OPERA *::=* *NONE* *NO-SEMANT* *;*
 *::=* OPERAND *NO-SEMANT* *;*
 *::=* OPERA OPERAND *NO-SEMANT* *;*
OPERAND *::=* STRING TYPE-OP *;*
 *::=* STRING TYPE-OP */* TYPE-OP *;*
TYPE-OP *::=* CONTROL CLASS TYPE *;*
CONTROL *::=* INTEGER *NO-SEMANT* *;*
        *::=* *#* *NO-SEMANT* *;*
CLASS *::=* PRIMITIVE *NO-SEMANT* *;*
 *::=* RESULT *NO-SEMANT* *;*
TYPE *::=* TYPE_A *NO-SEMANT* *;*
 *::=* TYPE_B *NO-SEMANT* *;*
 *::=* TYPE_C *NO-SEMANT* *;*
 *::=* TYPE_D *NO-SEMANT* *;*
 *::=* TYPE_E *NO-SEMANT* *;*
 *::=* TYPE_F *NO-SEMANT* *;*
 *::=* TYPE_G *NO-SEMANT*
*END-SYNTAX*
```

APPENDIX A -- CONTROL LANGUAGE SPECIFICATION

SEMANTICS

```
//GO.SEMANTIC DD *
 *SEMANTICS* SEMANT *CODE*
         DCL ASSIGN EXT CHAR(2000) VAR,
              (ARGPTR,MAX_PRIM,LENGTH_PRIM) EXT FIXED BIN;
         DCL LABEL INTERNAL ENTRY(FIXED BIN,CHAR(1)),
             INSERT INTERNAL ENTRY(CHAR(400) VAR);
 LABEL:  PROC(J,A);
         /*CREATES LABEL USING ANS */
         DCL J FIXED BIN,/*COL TO PRINT IN*/
             A CHAR(1),/*PRINT IMMEDIATELY AFTER LABEL*/
             (I,K) FIXED BIN;
             I=1;  K=10;
             DO WHILE (ANS>=K);
                I=I+1;   K=K*10;
                END;
             PUT FILE(OUT) EDIT('L',ANS,A)(COL(J),A,F(I),A);
             END LABEL;
 INSERT: PROC(A);
         /*INSERTS A AS AN ASSIGNMENT STATEMENT INTO ASSIGN*/
         DCL (A,B,C) CHAR(400) VAR,I FIXED BIN;
             IF A='' | A=' ' THEN RETURN;
             I=INDEX(A,'=');
             IF I=0 THEN RETURN;
             B=SUBSTR(A,1,I-1);
             C=SUBSTR(A,I+1);
             I=INDEX(ASSIGN,B)+LENGTH(B);
             ASSIGN=SUBSTR(ASSIGN,1,I)||C||SUBSTR(ASSIGN,I+3);
             END INSERT;
             *END*
*PRODUCTION* 1 *CODE*
        /*BUILD SLAVE RETURN OR INTERACTION LOOP END PROGRAM*/
        PUT FILE(OUT) EDIT(
            'EXIT1:       IF SLAVE THEN RETURN;',
            'ELSE GO TO L0;',
            'SAVE_PRIMITIVE:   ENTRY(A);',
            '/*REPLACES OR CREATES A NEW PRIMITIVE IF POSSIBLE*/',
            'ATEMP=FETCH_NAME(A);',
            'DO I=1 TO NUM_PRIM;',
            'IF ATEMP=FETCH_NAME(PRIMITIVES(I)) THEN DO;',
            'PRIMITIVES(I)=A;',
            'RETURN;',
            'END;',
            'END;',
            'IF NUM_PRIM<MAX_PRIM THEN DO;',
            'NUM_PRIM=NUM_PRIM+1;',
            'PRIMITIVES(NUM_PRIM)=A;',
            'RETURN;',
            'END;',
            'RETURN;')
            (COL(2),A,COL(18),A,COL(2),A,COL(10),A,
               2(COL(14),A),COL(18),A,3(COL(22),A),COL(18),A,
               COL(14),A,4(COL(18),A),COL(14),A);
        PUT FILE(OUT) EDIT (
            'INPUTT:   ENTRY(B);',
            '/*IF SLAVE MODE THEN INPUT FROM TEMP WITH BLANK',
```

```
    'OR QUOTES AS SEPARATOR',
    'ELSE CALL INPUT*/',
    'IF ¬ SLAVE THEN DO;',
    'CALL INPUT(B);',
    'RETURN;',
    'END;',
    'IF TEMP='''' | TEMP='' '' THEN DO;',
    'B='''';   RETURN;   END;',
    'DO WHILE(SUBSTR(TEMP,1,1)='' '');',
    'TEMP=SUBSTR(TEMP,2);',
    'END;',
    'I=INDEX(TEMP,'' '');',
    'J=INDEX(TEMP,QUOTES);',
    'IF I=0 & J=0 THEN DO;',
    'B=TEMP;',
    'TEMP='''';',
    'RETURN;',
    'END;',
    'IF J=1 THEN DO;',
    'TEMP=SUBSTR(TEMP,LENGTH(QUOTES)+1);',
    'J=INDEX(TEMP,QUOTES);',
    'IF J=0 THEN DO;',
    'B=SUBSTR(TEMP,1,LENGTH(TEMP)-2);',
    'TEMP='''';',
    'RETURN;',
    'END;',
    'B=SUBSTR(TEMP,1,J-1);',
    'TEMP=SUBSTR(TEMP,J+LENGTH(QUOTES));',
    'RETURN;',
    'END;',
    'IF I=0 THEN DO;',
    'B=SUBSTR(TEMP,1,J-1);',
    'TEMP=SUBSTR(TEMP,J);',
    'RETURN;',
    'END;',
    'B=SUBSTR(TEMP,1,I-1);',
    'TEMP=SUBSTR(TEMP,I+1);',
    'RETURN;')
    (COL(2),A,3(COL(10),A),COL(14),A,3(COL(18),A),
    COL(14),A,COL(18),A,
              COL(14),A,2(COL(18),A),3(COL(14),A),
    4(COL(18),A),COL(14),A,3(COL(18),A),4(COL(22),A),
    4(COL(18),A),COL(14),A,4(COL(18),A),3(COL(14),A));
PUT FILE(OUT) EDIT(
    'OUTPUTT:  ENTRY(B);',
    '/*DELETES OUTPUT ON SLAVE MODE*/',
    'IF ¬ SLAVE THEN CALL OUTPUT(B);',
    'RETURN;')
    (COL(2),A,COL(10),A,2(COL(14),A));
PUT FILE(OUT) EDIT(
    'DISPLAY_PRIMITIVES:  ENTRY(B);',
    '/*DISPLAYS THE PRIMITIVE NAMES WITH TITLE B LEVEL 2*/',
    'ATEMP=B||'' '';',
    'DO I=1 TO NUM_PRIM;',
    'ATEMP=ATEMP||FETCH_NAME(PRIMITIVES(I))||'' '';',
    'END;',
    'CALL INITIALIZE(2);',
    'CALL OUTPUTT(ATEMP);',
    'RETURN;')
    (COL(2),A,COL(10),A,2(COL(14),A),2(COL(18),A),3(COL(14),A));
```

```
        PUT FILE(OUT) EDIT(
            'DISPLAY_RESULT:   ENTRY(B);',
            '/*DISPLAYS RES_STACK(10) WITH TITLE B LEVEL 3*/',
            'ATEMP=B;',
            'IF ATEMP¬='' '' & ATEMP¬= '''' THEN',
            'DO WHILE(SUBSTR(ATEMP,1,1)='' '');',
            'ATEMP=SUBSTR(ATEMP,2);',
            'END;',
            'ATEMP=ATEMP||'' ''||RES_STACK(10);',
            'RESULT_NAME=SUBSTR(ATEMP,1,INDEX(ATEMP,'' '')-1);',
            'CALL INITIALIZE(3);',
            'CALL OUTPUTT(ATEMP);',
            'RETURN;')
            (COL(2),A,COL(10),A,2(COL(14),A),COL(18),A,2(COL(22),A),
            5(COL(14),A));
        PUT FILE(OUT) EDIT(
            'FIND_PRIMITIVE:    ENTRY(A,B);',
            '/*RETURNS IN B THE PRIMITIVE WHOSE NAME IS CONTAINED',
            '  IN A, NULL IF NO PRIMITIVE BY THAT NAME */',
            'DO I=1 TO NUM_PRIM;',
            'IF A=FETCH_NAME(PRIMITIVES(I)) THEN DO;',
            'B=PRIMITIVES(I); RETURN; END;',
            'END;',
            'B='''';  RETURN;',
            'EXIT2:   END INTERACTIVE;')
            (COL(2),A,2(COL(10),A),COL(14),A,COL(18),A,COL(22),A,
            COL(18),A,COL(14),A,COL(2),A);
            *END*
*PRODUCTION* 2 *CODE*
        /*OUTPUT PROCEDURE DECLS,VARIABLE DECLS,SLAVE AND
          INTERACTIVE ENTRY, INITIALIZATION*/
        PUT FILE(OUT) EDIT(
            'INTERACTIVE:   PROC OPTIONS(MAIN) RECURSIVE;',
            'DCL A CHAR(*) VAR, /* ENTRY PARAMETER */',
            'ARG(5) EXT CHAR(2000) VAR, /* OPERAND AREA */',
            'ASEP EXT CHAR(20) VAR, /* ATTRIBUTE SEPARATOR */',
            'ATEMP CHAR(2000) VAR, /* INPUT BUFFER */',
            'B CHAR(*) VAR, /* ENTRY PARAMETER */',
            'CCNNECTOR EXT CHAR(20) VAR, /* ATTR VALUE CONNECTOR */',
            'CSEP EXT CHAR(20) VAR, /* CONSTRUCTION SEPARATOR */',
            'I FIXED BIN, /* LOCAL VARIABLE */',
            'IN CHAR(*) VAR, /* ENTRY PARAMETER */',
            'J FIXED BIN, /* LOCAL VARIABLE */',
            'LEFT_PAREN EXT CHAR(20) VAR, /* LEFT PARENTHESIS */',
            'MARKER EXT CHAR(20) VAR, /* OPERAND SEPARATOR */',
            'MAX_PRIM EXT FIXED BIN, /* MAX NO OF PRIMITIVES */',
            'NUM_PRIM EXT FIXED BIN, /* ACTUAL NO OF PRIMS   */',
            'NUM_UNARY EXT FIXED BIN, /* NO OF UNARY OPS   */',
            'QUOTES EXT CHAR(20) VAR, /* QUOTES */',
            'RES_STACK(10) EXT CHAR(200) VAR, /*RESULT STACK */',
            'RESULT_NAME EXT CHAR(100) VAR,/*FIRST WORD OF RES TITLE*/',
            'RIGHT_PAREN EXT CHAR(20) VAR, /* RIGHT PARENTHESIS */',
            'SEP EXT CHAR(20) VAR, /* PRIMITIVE FIELD SEPARATOR */',
            'SLAVE EXT BIT(1), /* SLAVE SWITCH */',
            'TEMP EXT CHAR(2000) VAR, /* INPUT BUFFER */',
            'TERMINATOR EXT CHAR(20) VAR; /* TERMINATION CONTROL*/')
            (COL(2),A,COL(10),A,22(COL(14),A));
        PUT FILE(OUT) EDIT (
            'DCL (FETCH_ATTR,FETCH_BDF,FETCH_TS,FETCH_NAME,',
            'FETCH_PRIMITIVE) EXTERNAL ENTRY',
```

```
    '(CHAR(*) VAR) RETURNS(CHAR(2000) VAR),',
    'OPERAND INTERNAL ENTRY(FIXED BIN,CHAR(2000) VAR,',
    'CHAR(*) VAR,CHAR(*) VAR),',
    'UPDATE EXTERNAL ENTRY (CHAR(*) VAR),',
    'POP EXTERNAL ENTRY RETURNS(CHAR(2000) VAR),',
    'INITIALIZE EXTERNAL ENTRY(FIXED BIN);')
    (COL(10),A,2(COL(18),A),COL(14),A,COL(18),A,
          3(COL(14),A));
PUT FILE(OUT) EDIT(
    'DCL UNARY(',
    ANS,
    ') EXT CHAR(20) VAR,',
    'PRIMITIVES (',
    MAX_PRIM,
    ') EXT CHAR(',LENGTH_PRIM,') VAR;')
    (COL(10),A,F(4),A,COL(14),A,F(4),A,F(4),A);
PUT FILE(OUT) EDIT( ,
    'VAR_INIT:  PROC;',
    'DCL I FIXED BIN;',
    '/*INITIALIZES EXTERNAL VARIABLES*/',
    'DO I=1 TO 10; RES_STACK(I)='''''; END;',
    'DO I=1 TO 5;  ARG(I)='''';  END;',
    'RESULT_NAME='''';',
    'NUM_UNARY=',ANS,';','NUM_PRIM=0;')
    (COL(2),A,2(COL(10),A),4(COL(14),A),F(4),A,COL(14),A);
/*OUTPUT ASSIGN*/
DO WHILE (ASSIGN¬='' & ASSIGN¬=' ');
     ANS=INDEX(ASSIGN,';');
     IF ANS<=LENGTH(ASSIGN)-2 & SUBSTR(ASSIGN,ANS+1,1)=''''
        THEN ANS=ANS+2;
     PUT FILE(OUT) EDIT(SUBSTR(ASSIGN,1,ANS))(COL(14),A);
     IF ANS=LENGTH(ASSIGN) THEN ASSIGN='';
       ELSE ASSIGN=SUBSTR(ASSIGN,ANS+1);
     END;
     ANS=0;
/*CALL DISPLAY OF FUNCTIONS AND CONSTRUCTIONS*/
PUT FILE(OUT) EDIT(
    'CALL FUNCT;',
    'CALL CONST;')
    (2(COL(14),A));
PUT FILE(OUT) EDIT('END VAR_INIT;')(COL(14),A);
PUT FILE(OUT) EDIT(
    'OPERAND:   PROC(I,MSG,A,B);',
    '/*FETCHES OPERAND LEAVES IN ARG(I)*/',
    'DCL I FIXED BIN,',
    'TYP CHAR(50) VAR,(A,B,MSG) CHAR(*) VAR,',
    '(C,D) CHAR(2000) VAR,',
    '(AA(3),BB(3)) CHAR(20) VAR,',
    '(K,L) FIXED BIN;')
    (COL(2),A,2(COL(10),A),4(COL(14),A));
PUT FILE(OUT) EDIT(
    'DCL TYPE INTERNAL ENTRY(CHAR(*) VAR)',
    'RETURNS(CHAR(50) VAR);',
    'TYPE:   PROC(A)  RETURNS(CHAR(50) VAR);',
    'DCL I FIXED BIN,B CHAR(2000) VAR,',
    'A CHAR(*) VAR;',
    '/*RETURNS TYPE OF A*/',
    'IF A=RESULT_NAME THEN DO;',
    'A=RES_STACK(10);',
    'RETURN(''RESULT'');',
```

```
     'END;',
     'DO I=1 TO NUM_UNARY;',
     'IF A=UNARY(I) THEN RETURN(''UNARY'');',
     'END;',
     'B=FETCH_PRIMITIVE(A);',
     'IF B='''' THEN RETURN('''');',
     'ELSE RETURN(''PRIMITIVE'');',
     'END TYPE;')
     (COL(10),A,COL(14),A,COL(2),A,3(COL(10),A),COL(14),A
     ,3(COL(18),A),COL(14),A,2(COL(18),A),2(COL(14),A),COL(18),
     A,COL(14),A);
PUT FILE(OUT) EDIT (
     'CONSTRUCT:  PROC(A,B);',
     '/*LEAVES OPERAND IN ARG(I)*/',
     'DCL (A,B) CHAR(20) VAR,',
     '(J,K,L) FIXED BIN;',
     'MERGE1:   PROC;',
     '/*MERGES UNARY FOR TYPE A,B,C*/',
     'DCL B CHAR(20) VAR;',
     'DO K=1 TO L;',
     'IF C='''' | C='' '' THEN RETURN;',
     'J=INDEX(C,'' '');',
     'B=SUBSTR(C,1,J-1);',
     'C=SUBSTR(C,J+1);',
     'ARG(I)=B||MARKER||ARG(I);',
     'END;',
     'END MERGE1;')
     (COL(2),A,3(COL(10),A),COL(2),A,2(COL(10),A),COL(14),A,
     6(COL(18),A),COL(14),A);
PUT FILE(OUT) EDIT(
     'MERGE2:   PROC;',
     '/*MERGES UNARY OPERATORS FOR TYPE D AND E*/',
     'DCL B CHAR(20) VAR;',
     'DO K=1 TO L;',
     'IF C='''' | C='' '' THEN RETURN;',
     'J=INDEX(C,'' '');',
     'B=SUBSTR(C,1,J-1);',
     'C=SUBSTR(C,J+1);',
     'ARG(I)=LEFT_PAREN||MARKER||B||MARKER||ARG(I)||MARKER||',
     'RIGHT_PAREN;',
     'END;',
     'END MERGE2;')
     (COL(2),A,2(COL(10),A),COL(14),A,6(COL(18),A),COL(22),A,
     COL(14),A);
PUT FILE(OUT) EDIT(
     'MERGE3:   PROC;',
     '/*MERGES UNARY OPERATORS FOR TYPE F AND G*/',
     'DCL B CHAR(20) VAR;',
     'DO K=1 TO L;',
     'IF C='''' | C='' '' THEN RETURN;',
     'J=INDEX(C,'' '');',
     'B=SUBSTR(C,1,J-1);',
     'C=SUBSTR(C,J+1);',
     'ARG(I)=B||MARKER||LEFT_PAREN||MARKER||ARG(I)||MARKER||',
     'RIGHT_PAREN;',
     'END;',
     'END MERGE3;')
     (COL(2),A,2(COL(10),A),COL(14),A,6(COL(18),A),COL(22),A,
     COL(14),A);
PUT FILE(OUT) EDIT(
```

```
     'ARG(I)=D;',
     'IF A=''***'' THEN L=32000; ELSE L=A;',
     'IF B=''TYPE_A'' THEN DO;',
     'CALL MERGE1;',
     'RETURN;',
     'END;',
     'IF B=''TYPE_B'' THEN DO;',
     'ARG(I)=LEFT_PAREN||MARKER||ARG(I)||MARKER||RIGHT_PAREN;',
     'CALL MERGE1;',
     'RETURN;',
     'END;',
     'IF B=''TYPE_C'' THEN DO;',
     'CALL MERGE1;',
     'ARG(I)=LEFT_PAREN||MARKER||ARG(I)||MARKER||RIGHT_PAREN;',
     'RETURN;',
     'END;',
     'IF B=''TYPE_D'' THEN DO;',
     'ARG(I)=LEFT_PAREN||MARKER||ARG(I)||MARKER||RIGHT_PAREN;',
     'CALL MERGE2;',
     'RETURN;',
     'END;',
     'IF B=''TYPE_E'' THEN DO;',
     'CALL MERGE2;',
     'RETURN;',
     'END;',
     'IF B=''TYPE_F'' THEN DO;',
     'CALL MERGE3;',
     'RETURN;',
     'END;',
     'IF B=''TYPE_G'' THEN DO;',
     'CALL MERGE3;',
     'IF INDEX(ARG(I),LEFT_PAREN) ¬= 0 THEN',
     'ARG(I)=LEFT_PAREN||MARKER||ARG(I)||MARKER||',
     'RIGHT_PAREN;','RETURN;',
     'END;',
     'END CONSTRUCT;')
     (3(COL(14),A),3(COL(18),A),3(COL(14),A,4(COL(18),A)),
     2(COL(14),A,3(COL(18),A)),COL(14),A,2(COL(18),A),
     2(COL(22),A),3(COL(18),A));
PUT FILE (OUT) EDIT(
     'ARG(I)='''''; C='''';',
     '/*UNPACK A AND B */',
     'DO K=1 TO 2;',
     'L=INDEX(A,'' '');',
     'AA(K)=SUBSTR(A,1,L-1);',
     'A=SUBSTR(A,L+1);',
     'END;',
     'AA(3)=A;',
     'IF B='''' THEN DO K=1 TO 3;',
     'BB(K)='''';',
     'END;',
     'ELSE DO;',
     'DO K=1 TO 2;',
     'L=INDEX(B,'' '');',
     'BB(K)=SUBSTR(B,1,L-1);',
     'B=SUBSTR(B,L+1);',
     'END;',
     'BB(3)=B;',
     'END;')
     (3(COL(14),A),4(COL(18),A),2(COL(14),A),2(COL(18),A),
```

```
            COL(14),A,COL(18),A,4(COL(22),A),2(COL(18),A));
        PUT FILE(OUT) EDIT(
            'LOOP:   CALL INITIALIZE(8);',
            'CALL OUTPUTT(MSG);',
            'CALL INPUTT(D);',
            'IF D='''' THEN RETURN;',
            'TYP=TYPE(D);',
            'IF TYP=AA(2) THEN DO;',
            '/*OPTION ONE TERMINATED */',
            'C=C||''  '';',
            'CALL CONSTRUCT(AA(1),AA(3));',
            'RETURN;',
            'END;',
            'ELSE IF TYP=BB(2) THEN DO;',
            '/*OPTION TWO TERMINATED */',
            'C=C||''  '';',
            'CALL CONSTRUCT(BB(1),BB(3));',
            'RETURN;',
            'END;',
            'ELSE IF TYP=''UNARY'' THEN C=D||'' ''||C;',
            'GO TO LOOP;',
            'END OPERAND;')
            (COL(2),A,5(COL(14),A),5(COL(18),A),COL(14),A,5(COL(18),A),
            3(COL(14),A));
        PUT FILE(OUT) EDIT(
            'SLAVE=''0''B;',
            'CALL SETUPI;',
            'CALL SETUPO;',
            'CALL VAR_INIT;',
            'GO TO LO;',
            'SLAVES:     ENTRY(IN);',
            'SLAVE=''1''B;',
            'TEMP=IN||''  '';',
            'IF TEMP=''*INIT*'' THEN DO;',
            'CALL VAR_INIT;',
            'RETURN;',
            'END;',
            'LO:   CALL INITIALIZE(7);',
            'CALL OUTPUTT(''''||VS(J+1)||'''');',
            'CALL INPUTT(ATEMP);')
            (5(COL(14),A),COL(2),A,3(COL(14),A),3(COL(18),A),
            COL(2),A,2(COL(14),A));
            *END*
*PRODUCTION* 3 *CODE*
        /*INITIALIZE MAX_PRIM */
        MAX_PRIM=1;  LENGTH_PRIM=1;
        *END*
*PRODUCTION* 5 *CODE*
        /*INITIALIZE ANS TO COUNT UNARY OPERATORS
          PUT ASSIGNMENT STMT INTO ASSIGN */
        ANS=1;
        ASSIGN=ASSIGN||'UNARY(1)='''||VS(J)||''';';
        *END*
*PRODUCTION* 6 *CODE*
        /*UPDATE ANS PUT ASSIGNMENT STMT INTO ASSIGN */
        ANS=ANS+1;
        ASSIGN=ASSIGN||'UNARY('||ANS||')='''||VS(K)||''';';
        *END*
*PRODUCTION* 8 *CODE*
        /*INITIALIZE ASSIGN AND MAX_PRIM, INSERT ITEM VALUE*/
```

```
        ASSIGN='SEP='''';QUOTES='''';CONNECTOR='''';CSEP='''';'||
               'LEFT_PAREN='''';MAX_PRIM=0 ;RIGHT_PAREN='''';'||
               'TERMINATOR='''';MARKER='''';TEMP='''';'||
               'ASEP='''';';
        CALL INSERT(VS(J));
        *END*
*PRODUCTION* 9 *CODE*
        /*INSERT ITEM VALUE IN ASSIGN */
        CALL INSERT(VS(K));
        *END*
*PRODUCTION* 10 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 11 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 12 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 13 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 14 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 15 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 16 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 17 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 18 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K) IN PROPER FORM */
        VS(J)=VS(J)||'='||''''||VS(K)||'''';
        *END*
*PRODUCTION* 19 *CODE*
        /*SAVE VS(J) VS(J+1) VS(K) IN PROPER FORM*/
        /*SAVE VALUE FOR PRIMITIVE DECL */
        VS(J)=VS(J)||'='||VS(K);
        MAX_PRIM=VS(K);
        *END*
*PRODUCTION* 20 *CODE*
        /* SAVE LENGTH OF PRIMITIVE FOR DECLARATION*/
        LENGTH_PRIM=VS(K);
        IF LENGTH_PRIM>2000 THEN LENGTH_PRIM=2000;
        VS(J)='';
        *END*
*PRODUCTION* 21 *CODE*
        /*ISSUE DUMMY CONST PROCEDURE AND CALL FUNCTION*/
```

```
        PUT FILE(OUT) EDIT(
            'CONST:    PROC;',
            '/*DUMMY PROCEDURE*/',
            'END CONST;')(COL(2),A,COL(10),A,COL(14),A);
        PUT FILE(OUT) EDIT('CALL FUNCTION;')(COL(14),A);
        *END*
*PRODUCTION* 22 *CODE*
        /*ISSUE CALL FUNCTION AND CALL CONSTRUCTION*/
        PUT FILE(OUT) EDIT(
            'CALL FUNCTION;',
            'CALL CONSTRUCTION;')
            (2(COL(14),A));
            *END*
*PRODUCTION* 23 *CODE*
        /*END PROCEDURE, OUTPUT FUNCT PROCEDURE*/
        PUT FILE(OUT) EDIT(
            'END FUNCTION;',
            'FUNCT:    PROC;',
            '/*OUTPUTS FUNCTION NAMES*/',
            'ARG(1)=')
            (COL(14),A,COL(2),A,COL(10),A,COL(14),A);
        VS(J)=VS(J)||' '||VS(J+1)||' ';
        DO WHILE(LENGTH(VS(J))>0);
           IF LENGTH(VS(J))>50 THEN DO;
              PUT FILE(OUT) EDIT(
                  '''',SUBSTR(VS(J),1,50),'''||')
                  (COL(19),3 A);
                  VS(J)=SUBSTR(VS(J),51);
                  END;
              ELSE DO;
                PUT FILE(OUT) EDIT('''',VS(J),''';')(COL(19),3 A);
                VS(J)='';
                END;
           END;
        PUT FILE(OUT) EDIT(
            'CALL INITIALIZE(1);',
            'CALL OUTPUTT(ARG(1));',
            'END FUNCT;')
            (3(COL(14),A));
            *END*
*PRODUCTION* 24 *CODE*
         /*SAVE FUNCTION CLASS NAME FOR FUNCT*/
         /*ISSUE BEGINNING OF FUNCTION PROCEDURE*/
         VS(J)=VS(K);
         PUT FILE(OUT) EDIT(
             'FUNCTION:   PROC;',
             '/*TESTS FOR FUNCTION TYPE PROCEDURES*/')
             (COL(2),A,COL(10),A);
             *END*
*PRODUCTION* 25 *CODE*
         /*END PROCEDURE, OUTPUT CONST PROCEDURE*/
         PUT FILE(OUT) EDIT(
             'END CONSTRUCTION;',
             'CONST:    PROC;',
             '/*OUTPUTS CONSTRUCTION NAMES*/',
             'ARG(1)=')
             (COL(14),A,COL(2),A,COL(10),A,COL(14),A);
         VS(J)=VS(J)||' '||VS(J+1)||' ';
         DO WHILE(LENGTH(VS(J))>0);
            IF LENGTH(VS(J))>50 THEN DO;
```

```
                PUT FILE(OUT) EDIT(
                    '''',SUBSTR(VS(J),1,50),'''||')
                    (COL(19),3 A);
                    VS(J)=SUBSTR(VS(J),51);
                    END;
                ELSE DO;
                  PUT FILE(OUT) EDIT('''',VS(J),''';')(COL(19),3 A);
                  VS(J)='';
                  END;
            END;
        PUT FILE(OUT) EDIT(
            'CALL OUTPUTT(ARG(1));',
            'END CONST;')
            (2(COL(14),A));
            *END*
*PRODUCTION* 26 *CODE*
         /*SAVE CONSTRUCTION CLASS NAME FOR CONST*/
         /*ISSUE BEGINNING OF CONSTRUCTION PROCEDURE*/
         VS(J)=VS(K);
         PUT FILE(OUT) EDIT(
             'CONSTRUCTION:   PROC;',
             '/*TESTS FOR CONSTRUCTION TYPE PROCEDURES*/')
             (COL(2),A,COL(10),A);
             *END*
*PRODUCTION* 28 *CODE*
        /*INSERT END FOR THIS OP-COMMAND*/
        PUT FILE(OUT) EDIT(
            'END ')(COL(14),A);
        CALL LABEL(18,';');
        *END*
*PRODUCTION* 29 *CODE*
        /*CONCATENATE OPERATION NAMES */
        /*INSERT END FOR OP-COMMAND*/
        PUT FILE(OUT) EDIT(
            'END ')(COL(14),A);
            CALL LABEL(18,';');
            VS(J)=VS(J)||' '||VS(K);
            *END*
*PRODUCTION* 31 *CODE*
        /*BUILD IF TEST FOR OPERATION AND LABEL DO
          UPDATE ANS   LEAVE WORD IN VS(J)   RESET ARGPTR */
        ARGPTR=0;
        PUT FILE(OUT) EDIT(
            'IF ATEMP='''||VS(K)||''' THEN')(COL(14),A);
        ANS=ANS+1;
        CALL LABEL(2,':');
        PUT FILE(OUT) EDIT(
            'DO;   /*OPTION '||VS(K)||'*/')(COL(14),A);
        VS(J)=VS(K);
        *END*
*PRODUCTION* 36 *CODE*
        /*ISSUE OPERAND CALL*/
        IF ARGPTR >= 5 THEN PUT FILE(DIAG) EDIT
            ('*****TOO MANY ARGS FOR THIS FUNCTION OR CONSTRUCTION')
             (SKIP,A);
            ELSE ARGPTR=ARGPTR+1;
        PUT FILE(OUT) EDIT(
            'CALL OPERAND(',ARGPTR,
            ',''J'||VS(J)||''',',
            '''''||VS(K)||''',''''');')
```

```
            (COL(18),A,F(4),2(COL(22),A));
            *END*
*PRODUCTION* 37 *CODE*
        /*ISSUE OPERAND CALL*/
        IF ARGPTR >= 5 THEN PUT FILE(DIAG) EDIT
           ('*****TOO MANY ARGS FOR THIS FUNCTION OR CONSTRUCTION')
           (SKIP,A);
           ELSE ARGPTR=ARGPTR+1;
        PUT FILE(OUT) EDIT(
            'CALL OPERAND(',ARGPTR,
            ','''||VS(J)||''',',
            '''''||VS(J+1)||''',',
            '''''||VS(K)||''');')
            (COL(18),A,F(4),3(COL(22),A));
            *END*
*PRODUCTION* 38 *CODE*
        /*SAVE VS(J) VS(J+1) AND VS(K)*/
        VS(J)=VS(J)||' '||VS(J+1)||' '||VS(K);
        *END*
*END-SEMANTICS*
```

APPENDIX B -- DEFINITION LANGUAGE SPECIFICATION

SYNTAX

```
//GO.SYNDATA DD *
 SYM(1)='*DEFINITION*' SYM(2)='PROG_NAME' SYM(3)='DLA' SYM(4)='SPEC1'
 SYM(5)='SPEC2' SYM(6)='SPEC3' ERRORSCAN='*END*' SEQUENCE='DEF-LANG'
 PARSER_NAME='DEF_LANG' SEMANT_NAME='SEMANT'   QUOTES=''''
 TERMINAL='*END-DEFINITION*' MLIM=50 NLIM=50 MMLIM=50;
/*
//GO.SYNTAX DD *
*SYNTAX*
DEF-LANG *::=* *DEFINITION* PROG_NAME DL *;*
PROG_NAME *::=* WORD   *;*
DL *::=* DLA SPEC1 SPEC2 SPEC3 *NO-SEMANT* *;*
 DLA *::=* *NAME* STRING *;*
SPEC1 *::=* *TS* *;*
 *::=* *TS* STRING *;*
SPEC2 *::=* *ATTR-LIST* *;*
 *::=* SPEC2A SPECA- *;*
SPEC2A *::=* *ATTR-LIST* STRING *;*
SPECA- *::=* SPECA *NO-SEMANT* *;*
SPECA *::=* SPEC *NO-SEMANT* *;*
 *::=* SPECA       SPEC *NO-SEMANT* *;*
SPEC *::=* WORD = NAME *;*
 *::=* WORD = TEXT   *;*
 *::=* WORD = NUMBER *;*
SPEC3 *::=* *BDF-FILE*  *;*
 *::=* SPEC3A SPECB- *;*
SPEC3A *::=* *BDF-FILE* STRING *;*
SPECB- *::=* SPECB *NO-SEMANT* *;*
SPECB *::=* CONSTRUCTION *NO-SEMANT* *;*
 *::=* SPECB *END* CONSTRUCTION *;*
CONSTRUCTION *::=* CONST PARM-LIS  *;*
 *::=* CONSTA PARM-LIS *;*
 CONST *::=* WORD INTEGER *;*
CONSTA *::=* WORD # *;*
PARM-LIS *::=* PARM-LIST *NO-SEMANT* *;*
PARM-LIST *::=* PARM-PAIR *NO-SEMANT* *;*
 *::=* PARM-LIST PARM-PAIR *NO-SEMANT* *;*
PARM-PAIR *::=* STRING NAME *;*
 *::=* STRING TEXT *;*
 *::=* STRING NUMBER
*END-SYNTAX*
```

APPENDIX B -- DEFINITION LANGUAGE SPECIFICATION

SEMANTICS

```
//GO.SEMANTIC DD *
*SEMANTICS*   SEMANT
   *CODE*  *END*
*PRODUCTION* 1 *CODE*
        /*OUTPUT END OF PROGRAM*/
        PUT FILE(OUT) EDIT(
            'END '||VS(J+1)||';')(COL(14),A);
            *END*
*PRODUCTION* 2 *CODE*
        /*OUTPUT PROC DECL PLUS DECLARATIONS AND INTERNAL PROCEDURES
          FOR CHECKING NAME AND NUMBER -- OUTPUT NAME OF PROCEDURE
          AND CODE FOR SLAVE MODE */
        PUT FILE(OUT) EDIT(
             VS(J)||':     PROC;',
            'DCL (A,B,C) CHAR(2000) VAR,I FIXED BIN,',
            'ASEP CHAR(20) EXT VAR,',
            'CONNECTOR CHAR(20) EXT VAR,',
            'TEMP EXT CHAR(2000) VAR,',
            'SLAVE EXT BIT(1),',
            'TERMINATOR CHAR(20) EXT VAR,',
            'SEP CHAR(20) EXT VAR,',
            'NAME INTERNAL ENTRY(CHAR(*) VAR) RETURNS(BIT(1)),',
            'NUMBER INTERNAL ENTRY(CHAR(*) VAR) RETURNS(BIT(1)),',
            'SAVE_PRIMITIVE EXTERNAL ENTRY(CHAR(*) VAR),',
            'OUTPUT EXTERNAL ENTRY(CHAR(*) VAR),',
            'INPUT EXTERNAL ENTRY(CHAR(*) VAR),',
            'INITIALIZE EXTERNAL ENTRY(FIXED BIN);')
            (COL(2),A,14(COL(10),A));
        PUT FILE(OUT) EDIT(
            'NAME:     PROC(A)  RETURNS(BIT(1)):',
            '/*TRUE IF A TYPE NAME*/',
            'DCL A CHAR(*) VAR,I FIXED BIN;',
            'IF A ='''' THEN RETURN(''0''B);',
            'IF SUBSTR(A,1,1)>=''A'' & SUBSTR(A,1,1)<=''Z''',
            '     & INDEX(A,'' '')=0 THEN DO I=2 TO LENGTH(A);',
            'IF SUBSTR(A,I,1)<''A'' THEN RETURN(''0''B);',
            'END;',
            'ELSE IF SUBSTR(A,1,1)>''Z'' & INDEX(A,'' '')=0 THEN',
            'DO I=2 TO LENGTH(A);',
            'IF SUBSTR(A,I,1)<=''Z'' THEN RETURN(''0''B);',
            'END;',
            'RETURN(''1''B);','END NAME;')
            (COL(2),A,2(COL(10),A),2(COL(14),A),3(COL(18),A),
            COL(14),A,COL(18),A,2(COL(22),A),2(COL(14),A));
        PUT FILE(OUT) EDIT(
            'NUMBER:     PROC(A)  RETURNS(BIT(1)):',
            '/*TRUE IF A IS OF TYPE NUMBER*/',
            'DCL A CHAR(*) VAR,X FLOAT BIN;',
            'ON CONVERSION GO TO FALSE;',
            'ON OVERFLOW GO TO FALSE;',
            'ON UNDERFLOW GO TO FALSE;',
            'X=A;',
            'RETURN(''1''B);',
            'FALSE:     RETURN(''0''B);',
            'END NUMBER;')
```

```
            (COL(2),A,2(COL(10),A),5(COL(14),A),COL(2),A,COL(14),A);
        PUT FILE(OUT) EDIT(
            'IF SLAVE   THEN DO;',
            'IF TEMP='''' | TEMP='' '' THEN RETURN;',
            'DO WHILE(SUBSTR(TEMP,1,1)='' '');',
            'TEMP=SUBSTR(TEMP,2);',
            'END;',
            'DO WHILE(SUBSTR(TEMP,LENGTH(TEMP),1)='' '');',
            'TEMP=SUBSTR(TEMP,1,LENGTH(TEMP)-1);',
            'END;',
            'CALL SAVE_PRIMITIVE(TEMP);',
            'RETURN;',
            'END;')
            (COL(14),A,2(COL(18),A),2(COL(22),A),COL(18),A,2(COL(22),A),
            3(COL(18),A));
        PUT FILE(OUT) EDIT(
            'CALL INITIALIZE(4);',
            'CALL OUTPUT('''||VS(J)||''');')
            (2(COL(14),A));
        *END*
*PRODUCTION* 4 *CODE*
        /*OUTPUT CODE FOR REQUESTING,OBTAINING AND CHECKING
          PRIMITIVE NAME AND BUILD PRIMITIVE */
        PUT FILE(OUT) EDIT(
            '/*REQUEST PRIMITIVE NAME*/',
            'B='''';',
            'DO WHILE(¬NAME(B));',
            'CALL INITIALIZE(5);',
            'CALL OUTPUT('''||VS(K)||''');',
            'CALL INPUT(B);',
            'END;',
            'C=B||SEP;')
            (3(COL(14),A),4(COL(18),A),COL(14),A);
            *END*
*PRODUCTION* 5 *CODE*
        /*NO TS--OUTPUT CODE TO CONCATENATE SEP TO PRIMITIVE*/
        PUT FILE(OUT) EDIT(
            'C=C||SEP;')(COL(14),A);
            *END*
*PRODUCTION* 6 *CODE*
        /*OUTPUT CODE FOR REQUESTING AND OBTAINING TS--BUILD
          PRIMITIVE */
        PUT FILE(OUT) EDIT(
            '/*REQUEST TS */',
            'CALL INITIALIZE(5);',
            'CALL OUTPUT('''||VS(K)||''');',
            'CALL INPUT(B);',
            'C=C||B||SEP;')
            (5(COL(14),A));
            *END*
*PRODUCTION* 7 *CODE*
        /*NO ATTR-LIST -- OUTPUT CODE TO CONCATENATE SEP TO PRIMITIVE*/
        PUT FILE(OUT) EDIT(
            'C=C||SEP;')(COL(14),A);
            *END*
*PRODUCTION* 8 *CODE*
        /*END OF ATTR-LIST -- OUTPUT CODE TO CANCATENATE SEP TO
          PRIMITIVE */
        PUT FILE(OUT) EDIT(
            'C=C||SEP;')(COL(14),A);
```

```
            *END*
*PRODUCTION* 9 *CODE*
        /* BEGINNING OF ATTR-LIST -- OUTPUT CODE TO OUTPUT TITLE */
        PUT FILE(OUT) EDIT(
            '/*REQUEST ATTRIBUTES*/',
            'CALL INITIALIZE(5);',
            'CALL OUTPUT('''||VS(K)||''');')
            (3(COL(14),A));
            *END*
*PRODUCTION* 13 *CODE*
        /*OUTPUT CODE TO REQUEST, OBTAIN AND CHECK ATTR VS(J) TYPE
          NAME.   BUILD PRIMITIVE */
        PUT FILE(OUT) EDIT(
            'B='';',
            'DO WHILE(¬NAME(B));',
            'CALL INITIALIZE(6);',
            'CALL OUTPUT( ''ENTER '||VS(J)||''');',
            'CALL INPUT(B);',
            'END;',
            'C=C||'''||VS(J)||'''||CONNECTOR||B||ASEP;')
            (2(COL(14),A),4(COL(18),A),COL(14),A);
            *END*
*PRODUCTION* 14 *CODE*
        /* OUTPUT CODE TO REQUEST AND OBTAIN ATTR VS(J) TYPE TEXT.
           BUILD PRIMITIVE */
        PUT FILE(OUT) EDIT(
            'CALL INITIALIZE(6);',
            'CALL OUTPUT( ''ENTER '  ||VS(J)||''');',
            'CALL INPUT(B);',
            'C=C||'''||VS(J)||'''||CONNECTOR||B||ASEP;')
            (4(COL(14),A));
            *END*
*PRODUCTION* 15 *CODE*
        /*OUTPUT CODE TO REQUEST, OBTAIN AND CHECK ATTR VS(J)
          TYPE NUMBER. BUILD PRIMITIVE */
        PUT FILE(OUT) EDIT(
            'B=''A'';',
            'DO WHILE(¬NUMBER(B));',
            'CALL INITIALIZE(6);',
            'CALL OUTPUT( ''ENTER '  ||VS(J)||''');',
            'CALL INPUT(B);',
            'END;',
            'C=C||'''||VS(J)||'''||CONNECTOR||B||ASEP;')
            (2(COL(14),A),4(COL(18),A),COL(14),A);
            *END*
*PRODUCTION* 16 *CODE*
         /* NO BDF-FILE--ISSUE CODE TO CONCATENATE SEP AND
            SAVE PRIMITIVE  */
         PUT FILE(OUT) EDIT (
             'C=C||SEP;',
             'CALL SAVE_PRIMITIVE(C);')
             (2(COL(14),A));
             *END*
*PRODUCTION* 17 *CODE*
        /*END OF BDF-FILE.  OUTPUT CODE TO LOOP ON CONSTRUCTION TYPE.
          OUTPUT OPTION PROCEDURE WHICH CONTAINS ALL THE CONSTRUCTION
          OPTIONS */
        PUT FILE(OUT) EDIT (
            'GO TO BDF_FILE;',
            'OPTION:    PROC;',
```

```
            '/*SETS B=BDF OPTIONS */',
            'B=')
            (COL(14),A,COL(2),A,COL(10),A,1(COL(14),A));
            DO WHILE (LENGTH(VS(K))>0);
               IF LENGTH(VS(K))>50 THEN DO;
                  PUT FILE(OUT) EDIT('''',SUBSTR(VS(K),1,50),''''||')
                      (COL(19),3 A);
                  VS(K)=SUBSTR(VS(K),51);
                  END;
               ELSE DO;
                  PUT FILE(OUT) EDIT('''',VS(K),' '';')(COL(19),3 A);
                  VS(K)='';
                  END;
               END;
            PUT FILE(OUT) EDIT('END OPTION;')(COL(14),A);
            *END*
*PRODUCTION* 18 *CODE*
         /*OUTPUT CODE TO REQUEST BDF-FILE, DISPLAY OPTIONS AND
           INPUT SELECTED OPTION*/
         PUT FILE(OUT) EDIT(
            '/* REQUEST BDF-FILE */',
            'BDF_FILE:     CALL INITIALIZE(5);',
            'CALL OUTPUT('''||VS(K)||''');',
            'CALL OPTION;',
            'CALL OUTPUT(B);',
            'CALL INPUT(B);')
            (COL(2),A,5(COL(14),A));
            *END*
*PRODUCTION* 21 *CODE*
         /*BUILD CONSTRUCTION OPTIONS*/
         VS(J)=VS(J)||' '||VS(K);
         *END*
*PRODUCTION* 22 *CODE*
         /*END OF INTEGER TYPE OPTION. OUTPUT CODE TO SAVE_PRIMITIVE
           AND EXIT */
         PUT FILE(OUT) EDIT(
            'CALL SAVE_PRIMITIVE(C);',
            'RETURN;',
            'END;','END;')
            (3(COL(22),A),COL(18),A);
            *END*
*PRODUCTION* 23 *CODE*
         /*END OF INDEFINITE REPEATING OPTION*/
         PUT FILE(OUT) EDIT(
            'END;','END;')(COL(22),A,COL(18),A);
            *END*
*PRODUCTION* 24 *CODE*
         /*START OF INTEGER OPTION.  OUTPUT CODE TO START THE OPTION*/
         PUT FILE(OUT) EDIT(
            '/* TYPE '||VS(J)||'*/',
            'IF B='''||VS(J)||''' THEN DO;',
            'C=C||'''||VS(J)||''';',
            'DO I=1 TO '||VS(K)||';')
            (2(COL(14),A),2(COL(18),A));
            *END*
*PRODUCTION* 25 *CODE*
         /*START OF INDEFINITE REPEATING OPTION.  OUTPUT CODE TO
           START THIS OPTION*/
         PUT FILE(OUT) EDIT(
            '/*TYPE '||VS(J)||'*/',
```

```
            'IF B='''||VS(J)||''' THEN DO;',
            'C=C||'''||VS(J)||''';',
            'DO WHILE(''1''B);',
            'CALL INITIALIZE(6);',
            'CALL OUTPUT(''CONTINUE'');',
            'CALL INPUT(B);',
            'IF B=TERMINATOR THEN DO;',
            'CALL SAVE_PRIMITIVE(C);',
            'RETURN;',
            'END;')
            (2(COL(14),A),2(COL(18),A),4(COL(22),A),3(COL(26),A));
            *END*
*PRODUCTION* 29 *CODE*
        /*OUTPUT CODE TO REQUEST, OBTAIN AND CHECK NAME TYPE PARAMETER*/
        PUT FILE(OUT) EDIT(
            '/*PARAMETER '||VS(J)||'*/',
            'B='''';',
            'DO WHILE(¬NAME(B));',
            'CALL INITIALIZE(6);',
            'CALL OUTPUT('''||VS(J)||''');',
            'CALL INPUT(B);',
            'END;',
            'C=C|| '' '' ||B;')
            (3(COL(22),A),4(COL(26),A),COL(22),A);
            *END*
*PRODUCTION* 30 *CODE*
        /*OUTPUT CODE TO REQUEST AND OBTAIN TEXT TYPE PARAMETER*/
        PUT FILE(OUT) EDIT(
            '/*PARAMETER '||VS(J)||'*/',
            'CALL INITIALIZE(6);',
            'CALL OUTPUT('''||VS(J)||''');',
            'CALL INPUT(B);',
            'C=C|| '' '' ||B;')
            (5(COL(22),A));
            *END*
*PRODUCTION* 31 *CODE*
        /*OUTPUT CODE TO REQUEST,OBTAIN AND CHECK NUMBER TYPE
          OPTION*/
        PUT FILE(OUT) EDIT(
            '/*PARAMETER '||VS(J)||'*/',
            'B=''A'';',
            'DO WHILE(¬NUMBER(B));',
            'CALL INITIALIZE(6);',
            'CALL OUTPUT('''||VS(J)||''');',
            'CALL INPUT(B);',
            'END;',
            'C=C|| '' ''  ||B;')
            (3(COL(22),A),4(COL(26),A),COL(22),A);
            *END*
*END-SEMANTICS*
```

APPENDIX C -- GRAPHIC LANGUAGE SPECIFICATION

SKELETON GRAPHIC PARSER

```
*PARSER*:  PROC (SINPUT,SOUTPUT,SDIAG,SEMANT);
        /*PARSER USING THE TABLES INSERTED BY THE SYNTAX PROGRAM      */
      DCL  SINPUT CHAR(*) VAR, /*INPUT STRING TO BE PARSED */
           SOUTPUT CHAR(*) VAR, /*OUTPUT STRING*/
           SDIAG CHAR(*) VAR, /*DIAGNOSTIC OUTPUT STRING */
           SEMANT  ENTRY;  /*SEMANTIC PROCEDURE */
        DCL (I,J,K,L,KK,I1,I2,I3) FIXED BIN,
           S(0:25) FIXED BINARY, /*PARSING STACK*/
           V(0:25) CHAR(10)  VAR, /* VALUE STACK */
           QUOTE BIT(1),  /*BOOLEAN FOR QUOTING BASIC SYMBOLS */
           SYM FIXED BIN,  /* NUMERICAL FORM OF ASSIGNED SYMBOL */
           SYMS CHAR(400) VAR, /*STRING FORM OF ASSIGNED SYMBOL */
           ERROR BIT(1) INITIAL('0'B), /*PASSED TO SEMANT*/
           DIDDLE BIT(1) INITIAL('0'B), /*IF TRUE MOD INPUT STRING*/
           ANS FIXED BIN INITIAL(0), /*PASSED TO SEMANT*/
           INPUT CHAR(2000) VAR, /*INPUT BUFFER*/
           OUTPUT CHAR(2000) VAR, /*OUTPUT BUFFER*/
           DIAG CHAR(2000) VAR, /*DIAGNOSTIC BUFFER*/
           MARKER EXT CHAR(20) VAR; /*USED IN BUILDING STRINGS*/
*INSERT*
        DCL LOOK INTERNAL ENTRY(CHAR(400) VAR,FIXED BIN,BIT(1),BIT(1));
 LOOK:  PROC(S,I,T,X);
        /*FREE FIELD READ PROCEDURE  T IS FALSE IF INTEGER ELSE TRUE*/
        /*SEPARATOR IS ALWAYS BLANK IF NOT QUOTED STRING THEN A
          SEPARATOR IS ANY SINGLE CHARACTER IN THE SYNTAX
          IF X TRUE THEN BLANKS REMOVED ELSE BLANKS LEFT */
 NEXT:  PROC  RETURNS(CHAR(1));
        /* GETS THE NEXT CHARACTER FROM INPUT*/
           IF I>LENGTH(INPUT) THEN DO:
              DIAG='INPUT USED';
              GO TO FINIS;
              END;
           RETURN(SUBSTR(INPUT,I,1));
           END NEXT;
 CON:   PROC;
        /*CONCATENATES SYM TO S AND INCREASED I */
           S=S ||SYM; I=I+1;
           END CON;
 SPEC:  PROC(A,B)  RETURNS(BIT(1));
        /* TRUE IF A IS NOT A SEPARATING CHARACTER*/
        DCL A CHAR(1), B BIT(1), J FIXED BIN;
           IF A=' ' | A=QUOTES THEN RETURN('0'B);
           IF B THEN RETURN('1'B);
           IF MARKER ¬=' ' THEN RETURN('1'B);
           DO J=1 TO M;  IF A=BASSYM(J) THEN RETURN('0'B); END;
           RETURN('1'B);
           END SPEC;
        DCL SPEC INTERNAL ENTRY (CHAR(1),BIT(1)) RETURNS(BIT(1)),
            NEXT INTERNAL ENTRY RETURNS (CHAR(1)),
            CON INTERNAL ENTRY,
            SYM CHAR(1), CSYM CHAR(2) VAR, JJ FIXED BIN,
            (T,X) BIT(1),
            I FIXED BIN, /*INPUT BUFFER POINTER*/
            S CHAR(400) VAR; /*OUTPUT STRING*/
            SYM=NEXT;   S='';
```

```
            IF X THEN DO WHILE (SYM=' ');
               I=I+1;  SYM=NEXT;  END;
            IF ¬SPEC(SYM,QUOTE) THEN DO;
               CALL CON;  T='1'B;
               IF X THEN DO WHILE(NEXT=' ');
                  IF I=LENGTH(INPUT) THEN RETURN;
                   ELSE I=I+1;
                  END;
               CSYM=S||NEXT;
               SYM=NEXT;
               DO JJ=1 TO M;
                  IF CSYM=BASSYM(JJ) THEN CALL CON;
                  END;
               RETURN;
               END;
            IF SYM>'Z' THEN DO;
               DO WHILE (NEXT>'Z');
                  CALL CON;   SYM=NEXT;
                  END;
               T='0'B;  RETURN;
               END;
            DO WHILE (SPEC(SYM,QUOTE));
               CALL CON;   SYM=NEXT;
               END;
            T='1'B;   RETURN;
            END LOOK;
ASSIGN: PROC (QUOTE,OS,V)  RECURSIVE;
        /*ASSIGNS A NUMERICAL VALUE TO CURRENT INPUT SYMBOL */
        DCL QUOTE BIT(1),
            OS CHAR(400) VAR, /*STRING RETURNED HERE */
            V FIXED BIN,  /* NUMERICAL FORM OF STRING */
            J FIXED BIN,
            T BIT(1),OX CHAR(400) VAR;
            IF QUOTE THEN DO;
               CALL LOOK(OS,I,T,'0'B);
               IF OS=QUOTES THEN DO;
                  QUOTE='0'B;   OS='';  V=XSTRING;       RETURN;
                  END;
               CALL LOOK(OX,I,T,'0'B);
               DO WHILE(OX¬=QUOTES);
                  OS=OS||OX;
                  CALL LOOK(OX,I,T,'0'B);
                  END;
               QUOTE='0'B;  V=XSTRING;
               RETURN;
               END;
            CALL LOOK(OS,I,T,'1'B);
            IF T THEN DO;
               IF OS=QUOTES THEN DO;
                  QUOTE='1'B; CALL ASSIGN(QUOTE,OS,V);  RETURN;
                  END;
               DO J=1 TO M;
                  IF OS=BASSYM(J) THEN DO;
                     V=BASVAL(J); RETURN;
                     END;
                  END;
               IF OS=MARKER THEN DO;
                  CALL ASSIGN(QUOTE,OS,V);
                  RETURN;
                  END;
```

```
            V=XWORD;  RETURN;
            END;
        V=XINTEGER;  RETURN;
        END ASSIGN;
/* ---------     PARSING SECTION     ------------  */
DO J=0 TO 25;  S(J)=0;  V(J)= '';   END;
S(0)=XTERM;
I=1;                J=0;   QUOTE='0'B;
OUTPUT='';  DIAG='';
INPUT=SINPUT||' '||BASSYM(M)||' ';
CALL ASSIGN(QUOTE,SYMS,SYM);
DO WHILE (SYM>0);
  IF J>=25 THEN DO;
     DIAG='STACK OVERFLOW';
     GO TO FINIS;
     END;
  J=J+1;  K=J;  S(J)=SYM;  V(J)=SYMS;
  CALL ASSIGN(QUOTE,SYMS,SYM);
  DO WHILE (H(S(J),SYM)='>');
     IF S(J)=XSEQ THEN GO TO FINIS;
     DO WHILE ((H(S(J-1),S(J))='=') &(J>1));
        J=J-1;
        END;
     L=KEY(S(J));
     DO WHILE (PRTB(L)¬=0);
        KK=J+1;
        DO WHILE ((KK<=K) & (S(KK)=PRTB(L)));
           KK=KK+1;  L=L+1;
           END;
        IF ((KK>K) & (PRTB(L)<0)) THEN DO,
           I1=J;  I2=K;  I3=-PRTB(L);
           IF I3<=N THEN CALL  SEMANT (I3,V,I1,I2,ANS,ERROR,
                OUTPUT,DIAG,DIDDLE);
           S(J)=PRTB(L+1);   L=0;
           IF DIDDLE THEN DO;
              DIDDLE='0'B;
              IF LENGTH(INPUT)+3*LENGTH(MARKER)+LENGTH(DIAG)+
                  LENGTH(SYMS)>2000 THEN DO;
                    DIAG='INPUT OVERFLOW';
                    GO TO FINIS;
                    END;
              INPUT=SUBSTR(INPUT,1,I-1)||DIAG||
                    ' '||SYMS||' '||SUBSTR(INPUT,I);
              CALL ASSIGN(QUOTE,SYMS,SYM);
              DIAG='';
              J=J-1;
              END;
           END;
        ELSE DO;
           DO WHILE (PRTB(L)>0);
              L=L+1;
              END;
           L=L+2;
           END;
        END;
     IF L¬=0 THEN DO;        /*PUT ERROR RECOVERY HERE */
         DIAG='PARSING ERROR'||I||INPUT||J||K;
         DO J=0 TO K;  DIAG=DIAG||S(J)||V(J);  END;
         DIAG=DIAG||SYM||SYMS;
         GO TO FINIS;
         END;
     K=J;
     END;
  END;
FINIS:          SOUTPUT=OUTPUT;  SDIAG=DIAG;
  END *PARSER*;
*END*
```

APPENDIX C -- GRAPHIC LANGUAGE SPECIFICATION

GRAPHIC SEMANTIC CONSTRUCTOR SYNTAX

```
//GO.SYNDATA DD *
      PARSER_NAME='TS_LANG'  SEMANT_NAME='SEMANT'
/*
//GO.SYNTAX DD *
*SYNTAX*
SEMANTICS *::=* SEMANT  CODA  PRODUCTIONS *;*
PRODUCTIONS *::=* INTERPRETATIONS *NO-SEMANT* *;*
SEMANT *::=* *SEMANTICS* WORD *;*
INTERPRETATIONS *::=* INTERPRETATION *NO-SEMANT* *;*
*::=* INTERPRETATIONS INTERPRETATION *NO-SEMANT* *;*
INTERPRETATION *::=* INTERP *CODE* *;*
INTERP *::=* *PRODUCTION* INTEGER     *;*
CODA *::=* *CODE*
*END-SYNTAX*
```

APPENDIX C -- GRAPHIC LANGUAGE SPECIFICATION

GRAPHIC SEMANTIC CONSTRUCTOR SEMANTICS

```
//GO.SEMANTIC DD *
 *SEMANTICS*   SEMANT  *CODE*   *END*
 *PRODUCTION* 1 *CODE*
     PUT FILE(OUT) EDIT(
        'END '||VS(J)||';')(COL(10),A);
        CLOSE FILE(OUT);
        *END*
 *PRODUCTION* 3 *CODE*
     PUT FILE(OUT) EDIT(
        VS(J+1)||':  PROC(N,VS,LEFT,RIGHT,ANS,ERROR,OUT,DIAG,DIDDLE);',
        'DCL N FIXED BIN, /*CURRENT PRODUCTION NUMBER*/',
        'LEFT FIXED BIN, /*LEFT-HAND STACK POINTER */',
        'RIGHT FIXED BIN, /* RIGHT-HAND STACK POINTER */',
        'ANS FIXED BIN, /* INITIALLY SET TO ZERO */',
        'ERROR BIT(1), /* INITIALLY SET TO FALSE */',
        'OUT CHAR(2000) VAR, /* OUTPUT BUFFER */',
        'DIAG CHAR(2000) VAR, /* DIAGNOSTIC BUFFER */',
        'DIDDLE BIT(1), /*IF TRUE THEN MODIFY INPUT WITH V(J)*/',
        'VS(0:25) CHAR(10) VAR, /* VALUE STACK */',
        '(FETCH_ATTR,FETCH_BDF,FETCH_TS,FETCH_NAME,',
        'FETCH_PRIMITIVE) EXTERNAL ENTRY',
        '(CHAR(*) VAR) RETURNS(CHAR(2000) VAR),',
        'FETCH_VALUE EXT ENTRY(CHAR(*) VAR,CHAR(*) VAR)',
        'RETURNS(CHAR(2000) VAR);')
        (COL(2),A,COL(10),A,9(COL(14),A),2(COL(18),A),
        COL(14),A,COL(18),A);
     PUT FILE(OUT) EDIT(
        'DCL PTR FIXED BIN, /*POINTER TO LAST USED A & I */',
        'A(20) CHAR(20) VAR, I(20) FIXED BIN,',
        'R(20) FLOAT BIN,',
        '(CSEP,ASEP,LEFT_PAREN,RIGHT_PAREN,MARKER,CONNECTOR,',
        'SEP)  EXT CHAR(20) VAR,',
        'NEXT INT ENTRY(CHAR(*) VAR) RETURNS(CHAR(100) VAR),',
        'SAVE INT ENTRY(CHAR(*) VAR,CHAR(*) VAR)',
        'RETURNS(CHAR(2000) VAR),',
        'DELETE INT ENTRY(FIXED BIN) RETURNS (CHAR(20) VAR),',
        'FETCH INT ENTRY(CHAR(*) VAR,CHAR(*) VAR);')
        (COL(10),A,3(COL(14),A),COL(18),A,2(COL(14),A),
        COL(18),A,2(COL(14),A));
     PUT FILE(OUT) EDIT(
        'DELETE:    PROC(I)  RETURNS(CHAR(20) VAR);',
        '/*CONVERTS I TO CHAR STRING, DELETES BLANKS*/',
        'DCL I FIXED BIN, X CHAR(20) VAR;',
        'X=I;',
        'IF X='''' | X='' '' THEN RETURN('''');',
        'DO WHILE (SUBSTR(X,1,1)='' '');',
        'X=SUBSTR(X,2);',
        'END;',
        'DO WHILE (SUBSTR(X,LENGTH(X),1)='' '');',
        'X=SUBSTR(X,1,LENGTH(X)-1);',
        'END;',
        'RETURN(X);',
        'END DELETE;')
        (COL(2),A,2(COL(10),A),3(COL(14),A),2(COL(18),A),
        COL(14),A,2(COL(18),A),2(COL(14),A));
```

```
PUT FILE(OUT) EDIT(
   'NEXT:    PROC(NAMES)  RETURNS(CHAR(100) VAR);',
   '/*RETURNS THE NEXT NAME FROM NAMES DELETING IT FROM NAMES*/',
   'DCL NAMES CHAR(2000) VAR,NAME CHAR(100) VAR,J FIXED BIN;',
   'J=INDEX(NAMES,'' '');',
   'IF J=0 THEN DO;',
   'NAME=NAMES;',
   'NAMES='''';',
   'END;',
   'ELSE DO;',
   'NAME=SUBSTR(NAMES,1,J-1);',
   'NAMES=SUBSTR(NAMES,J+1);',
   'END;',
   'RETURN(NAME);',
   'END NEXT;')
   (COL(2),A,2(COL(10),A),2(COL(14),A),4(COL(14),A),
   3(COL(18),A),2(COL(14),A));
PUT FILE(OUT) EDIT(
   'FETCH:   PROC(PRIM,ANAMES);',
   '/*FETCHES THE VALUES OF THE NAMES IN ANAMES FROM ATTR',
   'IN PRIMITIVE PRIM, LEAVES ANS IN A,I AND R*/',
   'DCL (ATTR,NAMES) CHAR(2000) VAR,',
   '(ANAMES,PRIM) CHAR(*) VAR,',
   'NAME CHAR(100) VAR;',
   'ON UNDERFLOW ONSOURCE=''0'';',
   'ON OVERFLOW ONSOURCE=''0'';',
   'ON CONVERSION ONCHAR=''0'';',
   'NAMES=ANAMES;',
   'ATTR=FETCH_ATTR(PRIM);',
   'DO WHILE (NAMES ¬='''' & NAMES ¬='' '');',
   'NAME=NEXT(NAMES);',
   'IF PTR<20 THEN PTR=PTR+1;',
   'A(PTR)=FETCH_VALUE(ATTR,NAME);',
   'I(PTR)=A(PTR);',
   'R(PTR)=A(PTR);',
   'END;',
   'END FETCH;')
   (COL(2),A,3(COL(10),A),8(COL(14),A),6(COL(18),A),
   COL(14),A);
PUT FILE(OUT) EDIT(
   'SAVE:    PROC(ATTR,ANAMES)  RETURNS(CHAR(2000) VAR);',
   '/*SAVES VALUES OF ATTRIBUTE NAMES IN ANAMES FROM A',
   'INTO ATR, RETURNS ATR*/',
   'DCL (NAMES,ATR) CHAR(2000) VAR,',
   '(ATTR,ANAMES) CHAR(*) VAR,',
   'NAME CHAR(100) VAR;',
   'NAMES=ANAMES;',
   'ATR=ATTR;',
   'DO WHILE (NAMES¬='''' & NAMES¬='' '');',
   'NAME=NEXT(NAMES);',
   'IF PTR<20 THEN PTR=PTR+1;',
   'CALL SAVE_VALUE(ATR,NAME,A(PTR));',
   'END;',
   'RETURN(ATR);',
   'END SAVE;',
   'PTR=0;')
   (COL(2),A,3(COL(10),A),5(COL(14),A),4(COL(18),A),
   3(COL(14),A));
   VS(J)=VS(J+1);
   *END*
```

```
*PRODUCTION* 6 *CODE*
    PUT FILE(OUT) EDIT(
        'RETURN;',
        'END '||'L'||VS(J)||';')
        (2(COL(14),A));
        *END*
*PRODUCTION* 7 *CODE*
    PUT FILE(OUT) EDIT(
        'IF N=',
        VS(K),
        ' THEN',
        'L'||VS(K)||':',
        'DO;        /* PRODUCTION NUMBER ',
        VS(K),
        '*/')
        (COL(10),3 A,COL(2),A,COL(20),3 A);
        VS(J)=VS(K);
        *END*
*END-SEMANTICS*
```

APPENDIX D

GEM'S PROCEDURE LIBRARY

```
INITIALIZE:  PROC(N);
             /* INITIALIZES TO TYPE N */
             DCL (N,M) FIXED BIN,(BUFADD,BUFPTRS(8)) EXT FIXED BIN;
             DCL MINNZ INTERNAL ENTRY(FIXED BIN) RETURNS(FIXED BIN);
MINNZ:     PROC(A)  RETURNS(FIXED BIN);
           /*RETURNS FIRST NON-ZERO BUFPTRS(J) J>A */
           /* OR BUFADD IF NONE NON-ZERO FOR J>A */
           DCL (A,K) FIXED BIN;
              DO K=A+1 TO 8;
                 IF BUFPTRS(K)>0 THEN RETURN(BUFPTRS(K));
                 END;
              RETURN(BUFADD);
              END MINNZ;
                    IF N>8 THEN RETURN;
                    IF BUFADD<BUFPTRS(N) THEN BUFPTRS(N)=BUFADD;
                    ELSE IF BUFPTRS(N)=0 THEN
                         BUFPTRS(N),BUFADD=MINNZ(N);
                    ELSE BUFADD=BUFPTRS(N);
                    DO M=N+1 TO 8;
                       BUFPTRS(M)=0;
                       END;
              END INITIALIZE;


POP:       PROC  RETURNS(CHAR(2000) VAR);
           /*POPS UP RES_STACK RETURNING THE TOP ELEMENT */
           DCL A CHAR(2000) VAR, I FIXED BIN, RES_STACK(10) EXT
              CHAR(200) VAR;
              A=RES_STACK(10);
              DO I=10 TO 2 BY -1;
                 RES_STACK(I)=RES_STACK(I-1);
                 END;
              RES_STACK(1)='';
              RETURN(A);
              END POP;


UPDATE: PROC(A);
           /* UPDATES AND PUSHES DOWN THE RES_STACK */
           DCL A CHAR(*) VAR,I FIXED BIN,
              RES_STACK(10) EXT CHAR(200) VAR;
              DO I=1 TO 9;
                 RES_STACK(I)=RES_STACK(I+1);
                 END;
              RES_STACK(10)=A;
              END UPDATE;


FETCH_PRIMITIVE:   PROC(A)  RETURNS(CHAR(2000) VAR);
           /*FETCHES THE PRIMITIVE WHOSE NAME IS IN A */
           DCL (A,B) CHAR(2000) VAR,
               FIND_PRIMITIVE EXT ENTRY(CHAR(2000) VAR,
                          CHAR(2000) VAR);
               CALL FIND_PRIMITIVE(A,B);
               RETURN(B);
               END FETCH_PRIMITIVE;
```

```
SAVE_ATTR:  PROC(A,B);
        /*SAVE ATTR B IN PRIMITIVE A */
        DCL (A,B) CHAR(*) VAR,(I,J,K(3)) FIXED BIN,
           C CHAR(2000) VAR, SEP EXT CHAR(20) VAR;
SEARCH: PROC;
        /*LOCATES SEPS IN A */
           C=A;
           DO J=1 TO 3;
              I=INDEX(C,SEP);
              IF I=0 THEN K(J)=0;
                 ELSE DO;
                   K(J)=I;
                   IF SEP=' ' THEN SUBSTR(C,I,1)='.';
                     ELSE SUBSTR(C,I,1)=' ';
                   END;
              END;
           END SEARCH;
           CALL SEARCH;
           IF K(1)=0 THEN A=A||SEP||SEP||B||SEP;
             ELSE IF K(2)=0 THEN A=A||SEP||B||SEP;
             ELSE IF K(3)=0 THEN
                  A=SUBSTR(A,1,K(2)+LENGTH(SEP)-1)||B||SEP;
             ELSE A=SUBSTR(A,1,K(2)+LENGTH(SEP)-1)||B||SUBSTR(A,K(3));
           RETURN;
SAVE_BDF:  ENTRY(A,B);
        /*SAVE BDF B IN PRIMITIVE A*/
           CALL SEARCH;
           IF K(1)=0 THEN A=A||SEP||SEP||SEP||B;
             ELSE IF K(2)=0 THEN A=A||SEP||SEP||B;
             ELSE IF K(3)=0 THEN A=A||SEP||B;
             ELSE A=SUBSTR(A,1,K(3)+LENGTH(SEP)-1)||B;
           RETURN;
SAVE_TS:  ENTRY(A,B);
        /* SAVES TS FIELD B IN PRIMITIVE A */
           CALL SEARCH;
           IF K(1)=0 THEN A=A||SEP||B||SEP||SEP;
             ELSE IF K(2)=0 THEN
                A=SUBSTR(A,1,K(1)+LENGTH(SEP)-1)||B||SEP||SEP;
             ELSE IF K(3)=0 THEN
                A=SUBSTR(A,1,K(1)+LENGTH(SEP)-1)||B||SUBSTR(A,K(2))
                      ||SEP;
             ELSE A=SUBSTR(A,1,K(1)+LENGTH(SEP)-1)||B||SUBSTR(A,K(2));
           RETURN;
           END SAVE_ATTR;
```

```
SAVE_VALUE:  PROC(ATTR,NAME,VALUE);
        /*SAVES VALUE OF NAME IN ATTR */
        DCL (ATTR,NAME,VALUE) CHAR(*) VAR,(I,J,K) FIXED BIN,
           (CONNECTOR,ASEP) EXT CHAR(20) VAR;
           IF ATTR='' THEN DO;
               ATTR=NAME||CONNECTOR||VALUE;
               RETURN;
               END;
           I=INDEX(ATTR,NAME);
           IF I=0 THEN ATTR=ATTR||ASEP||NAME||CONNECTOR||VALUE;
             ELSE DO;
                K=LENGTH(ASEP);
                DO J=I+LENGTH(NAME) TO LENGTH(ATTR)-K;
                   IF SUBSTR(ATTR,J,K)=ASEP THEN DO;
                      ATTR=SUBSTR(ATTR,1,I-1)||NAME||CONNECTOR||
                           VALUE||ASEP||SUBSTR(ATTR,J+K);
                      RETURN;
                      END;
                   END;
                ATTR=SUBSTR(ATTR,1,I-1)||NAME||CONNECTOR||VALUE;
                END;
           END SAVE_VALUE;


FETCH_VALUE:  PROC(ATTR,NAME)  RETURNS(CHAR(2000) VAR);
        /* FETCHES THE VALUE OF NAME FROM ATTR */
        DCL (ATTR,NAME) CHAR(*) VAR,I FIXED BIN,A CHAR(2000) VAR,
           (CONNECTOR,ASEP) EXT CHAR(20) VAR;
           I=INDEX(ATTR,NAME);
           IF I=0 THEN RETURN('');
           A=SUBSTR(ATTR,I);
           I=INDEX(A,ASEP);
           IF I¬=0 THEN A=SUBSTR(A,1,I-1);
           I=INDEX(A,CONNECTOR);
           IF I=0 THEN RETURN('');
             ELSE RETURN(SUBSTR(A,I+LENGTH(CONNECTOR)));
           END FETCH_VALUE;
```

```
FETCH_NAME:  PROC(A)  RETURNS(CHAR(2000) VAR);
        /*RETURNS THE NAME FROM THE PRIMITIVE IN A  */
        DCL A CHAR(*) VAR, SEP EXT CHAR(20) VAR,
            K(3) FIXED BIN,(I,J) FIXED BIN;
SEARCH: PROC;
        /*LOCATES SEPS IN A */
        DCL C CHAR(2000) VAR;
            C=A;
            DO J=1 TO 3;
               K(J)=INDEX(C,SEP);
               IF K(J) ¬= 0 THEN DO;
                  IF SEP = ' ' THEN SUBSTR(C,K(J),1) = '.';
                     ELSE SUBSTR(C,K(J),1)=' ';
                  END;
               END;
            END SEARCH;
            CALL SEARCH;
            IF K(1)=0 THEN RETURN(A);
               ELSE IF K(1)=1 THEN RETURN('');
               ELSE RETURN(SUBSTR(A,1,K(1)-1));
FETCH_TS:   ENTRY(A) CHAR(2000) VAR;
        /*RETURNS THE TS FIELD FROM THE PRIMITIVE IN A */
            CALL SEARCH;
            I=K(2)-K(1)-LENGTH(SEP);
            IF I <= 0 THEN RETURN('');
               ELSE RETURN(SUBSTR(A,K(1)+LENGTH(SEP),I));
FETCH_ATTR:  ENTRY(A) CHAR(2000) VAR;
        /* RETURNS THE ATTRIBUTE FIELD FROM THE PRIMITIVE IN A */
            CALL SEARCH;
            I=K(3)-K(2)-LENGTH(SEP);
            IF I <= 0 THEN RETURN('');
               ELSE RETURN(SUBSTR(A,K(2)+LENGTH(SEP),I));
FETCH_BDF:   ENTRY(A) CHAR(2000) VAR;
        /* RETURNS THE BASIC DISPLAY FIELD FROM PRIMITIVE A */
            CALL SEARCH;
            I=LENGTH(A)-K(3)-LENGTH(SEP);
            IF I <= 0 THEN RETURN('');
               ELSE RETURN(SUBSTR(A,K(3)+LENGTH(SEP)));
            END FETCH_NAME;
```

APPENDIX E -- TWO-DIMENSIONAL MATHEMATICAL EXPRESSIONS

CONTROL DESCRIPTION

```
//GO.SOURCE DD *
 *CONTROL*    'SELECT OPTION OR OPERATOR'
 MAX_PRIM *=* 20
 RIGHT_PAREN *=* ')'
 LEFT_PAREN *=* '('
 CSEP *=* ':'
 CONNECTOR *=* '='
 QUOTES *=* '"'
 SEP *=* ';'
 MARKER *=* ''
 ASEP *=* ' '
 LENGTH_PRIM *=* 100
 -
 *CODE*
    DCL (SEMANT1,SEMANT2) EXT ENTRY, (BLK,XTR,YTR,CHAR_SIZE_X,
         CHAR_SIZE_Y,DIVS,BWIDTH,BHIGH,LUNDER,RUNDER,CDIV,NSPACE)
                 EXT FIXED BIN;
    DCL FIRST_ENTRY EXT BIT(1) INITIAL('1'B),XXTMP(11) FIXED BIN;
    DCL DBG EXT BIT(1) INITIAL('0'B);
SAVE_VAR:    PROC;
             /*SAVES AND SETS VARIABLES */
               XXTMP(1)=XTR;
               XXTMP(2)=YTR;
               XXTMP(3)=CHAR_SIZE_X;
               XXTMP(4)=CHAR_SIZE_Y;
               XXTMP(5)=DIVS;
               XXTMP(6)=BWIDTH;
               XXTMP(7)=BHIGH;
               XXTMP(8)=LUNDER;
               XXTMP(9)=RUNDER;
               XXTMP(10)=CDIV;
               XXTMP(11)=NSPACE;
               XTR=1;
               YTR=0;
               CHAR_SIZE_X=2;
               CHAR_SIZE_Y=1;
               DIVS=1;
               BWIDTH=1;
               BHIGH=1;
               LUNDER=0;
               RUNDER=-2;
               CDIV=2;
               NSPACE=1;
            END SAVE_VAR;
RESTORE_VAR:  PROC;
           /* RESTORES VARIABLES */
               XTR=XXTMP(1);
               YTR=XXTMP(2);
               CHAR_SIZE_X=XXTMP(3);
               CHAR_SIZE_Y=XXTMP(4);
               DIVS=XXTMP(5);
               BWIDTH=XXTMP(6);
               BHIGH=XXTMP(7);
               LUNDER=XXTMP(8);
               RUNDER=XXTMP(9);
```

```
                CDIV=XXTMP(10);
                NSPACE=XXTMP(11);
              END RESTORE_VAR;
     IF FIRST_ENTRY THEN DO;
        FIRST_ENTRY='0'B;
          BLK=1;  XTR=100;   YTR=300;  CHAR_SIZE_X=21;
          CHAR_SIZE_Y=30;   CDIV=1;  NSPACE=0;
          DIVS=4; BWIDTH=10; BHIGH=0; LUNDER=-11; RUNDER=-15;
        END;
          *END*
*FUNCTION*   'OPTIONS'
      *NAME* *=* ASSIGN  'SELECT TARGET'
           0 PRIMITIVE TYPE_A
           *CODE*
                ARG(2)=RES_STACK(10);
                ARG(1)=FETCH_PRIMITIVE(ARG(1));
                CALL SAVE_TS(ARG(1),ARG(2));
                ARG(2)='';
                CALL SAVE_ATTR(ARG(1),ARG(2));
                CALL SAVE_BDF(ARG(1),ARG(2));
                CALL SAVE_PRIMITIVE(ARG(1));
                *END*
      *NAME* *=* DEFINE *NONE*
           *CODE*
                CALL ANDDEF;
                CALL DISPLAY_PRIMITIVES('DEFINED VARIABLES ');
                CALL DISPLAY_RESULT('EXPRESSION = ');
                *END*
      *NAME*  *=* DISPLAY  *NONE*
           *CODE*
                ARG(2)='';
                ARG(1)=RES_STACK(10);
                CALL PARSERA(ARG(1),ARG(1),ARG(2),SEMANT1);
                ARG(2)=ARG(1)||' '||ARG(2);
                IF DBG THEN DO;
                   CALL INITIALIZE(4);
                   CALL OUTPUTT(ARG(2));
                    END;
                CALL INITIALIZE(5);
                CALL SCAN(ARG(1));
                *END*
      *NAME* *=* EVAL *NONE*
           *CODE*
                ARG(2)='';
                ARG(1)=RES_STACK(10);
                CALL PARSERA(ARG(1),ARG(1),ARG(2),SEMANT2);
                ARG(1)=ARG(1)||' '||ARG(2);
                CALL INITIALIZE(6);
                CALL OUTPUTT(ARG(1));
                *END*
      *NAME* *=* PRINT *NONE*
           *CODE*
                CALL SAVE_VAR;
                ARG(2)='';
                ARG(1)=RES_STACK(10);
                CALL PARSERA(ARG(1),ARG(1),ARG(2),SEMANT1);
                PUT EDIT(RES_STACK(10))(PAGE,COL(20),A);
                IF DBG THEN DO;
                   PUT EDIT('XTR=',XTR,'YTR=',YTR,'CHAR_SIZE_X=',
                         CHAR_SIZE_X,'CHAR_SIZE_Y=', CHAR_SIZE_Y,
```

```
                    'DIVS=',DIVS,'BWIDTH=',BWIDTH,'BHIGH=',
                    BHIGH,'LUNDER=',LUNDER,'RUNDER=',RUNDER,
                    'CDIV=',CDIV,'NSPACE=',NSPACE)
                    (SKIP(4),11(COL(20),A,F(20)));
               PUT EDIT(ARG(1),ARG(2))(SKIP(4),2(COL(20),A));
               END;
            CALL SCAN1(ARG(1));
            CALL RESTORE_VAR;
            CALL PRINTER;
            *END*
*NAME* *=* OUTPUT *NONE*
      *CODE*
            CALL SAVE_VAR;
            ARG(2)='';
            ARG(1)=RES_STACK(10);
            CALL PARSERA(ARG(1),ARG(1),ARG(2),SEMANT1);
            CALL SCAN1(ARG(1));
            CALL RESTORE_VAR;
            *END*
*NAME* *=* PRINTER *NONE*
      *CODE*
            CALL PRINTER;
            *END*
*NAME* *=* INIT  *NONE*
      *CODE*
            CALL INITIALIZE(4);
            CALL OUTPUTT('ERASE VARIABLES');
            CALL INPUTT(ATEMP);
            IF ATEMP='YES' THEN NUM_PRIM=0;
            BEGIN;
              ON CONVERSION ONSOURCE='0';
              CALL OUTPUTT('ENTER DBG');
              CALL INPUTT(ATEMP);
              DBG=ATEMP;
              CALL OUTPUTT('ENTER BLK');
              CALL INPUTT(ATEMP);
              BLK=ATEMP;
              CALL OUTPUTT('ENTER XTR');
              CALL INPUTT(ATEMP);
              XTR=ATEMP;
              CALL OUTPUTT('ENTER YTR');
              CALL INPUTT(ATEMP);
              YTR=ATEMP;
              CALL OUTPUTT('ENTER CHAR_SIZE_X');
              CALL INPUTT(ATEMP);
              CHAR_SIZE_X=ATEMP;
              CALL OUTPUTT('ENTER CHAR_SIZE_Y');
              CALL INPUTT(ATEMP);
              CHAR_SIZE_Y=ATEMP;
              CALL OUTPUTT('ENTER DIVS');
              CALL INPUTT(ATEMP);
              DIVS=ATEMP;
              CALL OUTPUTT('ENTER BWIDTH');
              CALL INPUTT(ATEMP);
              BWIDTH=ATEMP;
              CALL OUTPUTT('ENTER BHIGH');
              CALL INPUTT(ATEMP);
              BHIGH=ATEMP;
              CALL OUTPUTT('ENTER LUNDER');
              CALL INPUTT(ATEMP);
```

```
                LUNDER=ATEMP;
                CALL OUTPUTT('ENTER RUNDER');
                CALL INPUTT(ATEMP);
                RUNDER=ATEMP;
                CALL OUTPUTT('ENTER CDIV');
                CALL INPUTT(ATEMP);
                CDIV=ATEMP;
                CALL OUTPUTT('ENTER NSPACE');
                CALL INPUTT(ATEMP);
                NSPACE=ATEMP;
                END;
            CALL DISPLAY_PRIMITIVES('DEFINED VARIABLES ');
            CALL DISPLAY_RESULT('EXPRESSION = ');
            CALL INITIALIZE(4);
            ARG(1)='DBG'||DBG||' BLK'||BLK||' XTR'||XTR||' YTR'||
            YTR||' CHAR_SIZE_X'||CHAR_SIZE_X||' CHAR_SIZE_Y'||
                  CHAR_SIZE_Y||' DIVS'||DIVS||' BWIDTH'||BWIDTH||
                  ' BHIGH'||BHIGH||' LUNDER'||LUNDER||' RUNDER'||
                  RUNDER;
            CALL OUTPUTT(ARG(1));
            *END*
    *NAME* *=* INPUT *NONE*
        *CODE*
            CALL INITIALIZE(5);
            CALL OUTPUTT('ENTER EXPRESSION');
            CALL INPUTT(ARG(1));
            CALL UPDATE(ARG(1));
            CALL DISPLAY_RESULT('EXPRESSION = ');
            *END*
    *NAME* *=* POP *NONE*
        *CODE*
             ARG(1)=POP;
             CALL DISPLAY_RESULT('EXPRESSION = ');
             *END*
    *NAME* *=* TERM *NONE*
        *CODE*
            GO TO EXIT2;
            *END*
    *NAME* *=* DBG *NONE*
        *CODE*
            DBG=¬DBG;
            *END*
        *END*
*CONSTRUCTION*  'OPERATORS'
    *NAME* *=* + 'SELECT LEFT OPERAND'
        1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_F
        'SELECT RIGHT OPERAND'
        1 PRIMITIVE TYPE_E */* 1 RESULT TYPE_D
        *CODE*
             ARG(1)=ARG(1) || MARKER || '+' || MARKER || ARG(2);
             CALL UPDATE(ARG(1));
             CALL DISPLAY_RESULT('EXPRESSION = ');
             *END*
    *NAME* *=* - 'SELECT LEFT OPERAND'
        1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_F
        'SELECT RIGHT OPERAND'
        1 PRIMITIVE TYPE_E */* 1 RESULT TYPE_D
        *CODE*
             ARG(1)=ARG(1) || MARKER || '-' || MARKER || ARG(2);
             CALL UPDATE(ARG(1));
```

```
          CALL DISPLAY_RESULT('EXPRESSION = ');
          *END*
*NAME* *=* / 'SELECT DIVIDEND'
     1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
     'SELECT DIVISOR'
     1 PRIMITIVE TYPE_E */* 1 RESULT TYPE_D
     *CODE*
          ARG(1)=ARG(1) || MARKER || '/' || MARKER || ARG(2);
          CALL UPDATE(ARG(1));
          CALL DISPLAY_RESULT('EXPRESSION = ');
          *END*
*NAME* *=* * 'SELECT MULTIPLICAND'
     1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
     'SELECT MULTIPLIER'
     1 PRIMITIVE TYPE_E */* 1 RESULT TYPE_D
     *CODE*
          ARG(1)=ARG(1) || MARKER || '*' || MARKER || ARG(2);
          CALL UPDATE(ARG(1));
          CALL DISPLAY_RESULT('EXPRESSION = ');
          *END*
*NAME* *=* ** 'SELECT EXPRESSION'
     1 PRIMITIVE TYPE_E */* 1 RESULT TYPE_D
     'SELECT EXPONENT'
     1 PRIMITIVE TYPE_E */* 1 RESULT TYPE_D
     *CODE*
          ARG(1)=ARG(1) || MARKER || '**' || MARKER || ARG(2);
          CALL UPDATE(ARG(1));
          CALL DISPLAY_RESULT('EXPRESSION = ');
          *END*
*NAME* *=* NEG  *NONE*
     *CODE*
         ARG(1)='-'||MARKER||LEFT_PAREN||MARKER||
                RES_STACK(10)||MARKER||RIGHT_PAREN;
         CALL UPDATE(ARG(1));
         CALL DISPLAY_RESULT('EXPRESSION = ');
         *END*
     *END*
*END-CONTROL*
```

APPENDIX E -- TWO-DIMENSIONAL MATHEMATICAL EXPRESSIONS

DEFINITION DESCRIPTION

```
//GO.SOURCE DD *
 *DEFINITION*   ANDDEF
     *NAME*  'ENTER VARIABLE NAME'
     *TS*
     *ATTR-LIST*   'ENTER VALUE OF VARIABLE'
           VALUE=NUMBER
     *BDF-FILE*
     *END-DEFINITION*
/*
// EXEC PL1,PARM.PL1L='ATR,XREF,STMT'
//PL1L.SYSLIN DD DSNAME=WYL.CG.JEG.USERLIB(ANDDEF),DISP=(OLD,KEEP),
//  UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//PL1L.SYSIN DD DSNAME=&TEMP,DISP=(OLD,DELETE),UNIT=SYSDA
```

APPENDIX E -- TWO-DIMENSIONAL MATHEMATICAL EXPRESSIONS

SYNTAX DESCRIPTION

```
//GO.SYNDATA DD *
 SEQUENCE='EXPRESSION' PARSER_NAME='PARSERA' QUOTES=''''
   TERMINAL='END';
/*
//GO.SYNTAX DD *
 *SYNTAX*
EXPRESSION *::=* EXPR- *;*
EXPR- *::=* EXPR *NO-SEMANT* *;*
EXPR *::=* EXPR + TERM- *;*
 *::=* EXPR - TERM- *;*
 *::=* - TERM- *;*
 *::=* TERM- *NO-SEMANT* *;*
TERM- *::=* TERM *NO-SEMANT* *;*
TERM *::=* TERM * FACTOR- *;*
 *::=* TERM / FACTOR- *;*
 *::=* FACTOR- *NO-SEMANT* *;*
FACTOR- *::=* FACTOR *NO-SEMANT* *;*
FACTOR *::=* FACTOR ** PRIMARY *;*
       *::=* PRIMARY *NO-SEMANT* *;*
PRIMARY *::=* ( EXPR- ) *;*
 *::=* WORD *;*
 *::=* INTEGER
*END-SYNTAX*
```

APPENDIX E -- TWO-DIMENSIONAL MATHEMATICAL EXPRESSIONS

GRAPHICAL SEMANTIC DESCRIPTION

```
//GO.SOURCE DD *
*SEMANTICS*   SEMANT1     *CODE*
       DCL ARG(5) EXT CHAR(2000) VAR,STEMP CHAR(2000) VAR,
           BRACKET INTERNAL ENTRY(FIXED BIN),
           TRANSLATE INTERNAL ENTRY (CHAR(*) VAR,FIXED BIN,
           FIXED BIN) RETURNS(CHAR(2000) VAR),
           (CHAR_SIZE_X,CHAR_SIZE_Y) EXT FIXED BIN,
           (DIVS,BWIDTH,BHIGH,LUNDER,RUNDER) EXT FIXED BIN,
           MAKE_PRIMITIVE EXT ENTRY(CHAR(*) VAR)
           RETURNS(CHAR(2000) VAR);
TRANSLATE:  PROC(S,X,Y)  RETURNS(CHAR(2000) VAR);
            /* TRANSLATED BDF-FILE S TO X AND Y */
            DCL (X,Y) FIXED BIN, S CHAR(*) VAR, (A,B) CHAR(20) VAR;
               A=' ' || DELETE(X);
               B=' ' || DELETE(Y);
               RETURN('TRANS'||A||B||CSEP||S||CSEP||'UNTRAN'||A||B);
               END TRANSLATE;
BRACKET:  PROC(J);
          /*CREATES BRACKETS IN ARG(J)--RESETS PTR AND DESTROYS
             A(*),I(*),R(*) ,SETS UNARY=0 AND TYPE=0 */
          DCL J FIXED BIN;
          PTR=0;
             CALL FETCH(ARG(J),'LX RX BY TY MY');
             DO PTR=1 TO 5;
                 A(PTR)=' '||A(PTR);
                 END;
             A(6)=' '||DELETE(I(1)+BWIDTH);
             A(7)=' '||DELETE(I(2)+CHAR_SIZE_X-BWIDTH);
             A(8)=' '||DELETE(CHAR_SIZE_X+I(1));
             A(2)=' '||DELETE(I(2)+CHAR_SIZE_X);
             A(3)=' '||DELETE(I(3)-BHIGH);
             A(4)=' '||DELETE(I(4)+BHIGH);
             STEMP='LINE 0'||A(6)||A(4)||
                   ' 1'||A(1)||A(4)||
                   ' 1'||A(1)||A(3)||
                   ' 1'||A(6)||A(3)||
                   ' 0'||A(7)||A(4)||
                   ' 1'||A(2)||A(4)||
                   ' 1'||A(2)||A(3)||
                   ' 1'||A(7)||A(3)||
                   CSEP||'TRANS'||A(8)||' 0'||CSEP||FETCH_BDF(ARG(J))
                   ||CSEP||'UNTRAN'||A(8)||' 0';
             A(1)=DELETE(I(2)+2*CHAR_SIZE_X);  /*NEW RX*/
             A(2)=SUBSTR(A(4),2);              /*NEW TY */
             A(3)=SUBSTR(A(3),2);              /*NEW BY */
             A(4)='0';                         /*NEW TYPE */
             A(5)='0';                         /*NEW UNARY */
             PTR=0;
             CALL SAVE_ATTR(ARG(J),SAVE(FETCH_ATTR(ARG(J)),
                    'RX TY BY TYPE UNARY'));
             CALL SAVE_BDF(ARG(J),STEMP);
             PTR=0;
             END BRACKET;
MERGE:  PROC(S);
        /*MERGES BDFS FOR + - * AND NEW ATTR,SETS TYPE=S */
```

```
        DCL S CHAR(1);
            PTR=0;
            CALL FETCH(ARG(VS(LEFT)),'LX RX TY MY BY');
            CALL FETCH(ARG(VS(RIGHT)),'LX RX TY MY BY');
            STEMP=FETCH_BDF(ARG(VS(LEFT)))||CSEP||
                  TRANSLATE(STEMP,I(2),I(4));
            I(11)=I(2)+CHAR_SIZE_X-I(6); /*TRANS X */
            I(12)=I(4)-I(9);            /* TRANS Y */
            STEMP=STEMP||CSEP||TRANSLATE(FETCH_BDF(ARG(VS(RIGHT))),
                 I(11),I(12));
            CALL SAVE_BDF(ARG(VS(LEFT)),STEMP);
            A(1)=DELETE(I(11)+I(7)); /*NEW RX */
            A(2)=DELETE(MAX(I(3),I(8)+I(12)));  /*NEW TY*/
            A(3)=DELETE(MIN(I(5),I(10)+I(12)));  /*NEW BY*/
            A(4)=S;      /*NEW TYPE*/
            PTR=0;
            CALL SAVE_ATTR(ARG(VS(LEFT)),SAVE(FETCH_ATTR(ARG(VS(
                  LEFT))),   'RX TY BY TYPE'));
            ANS=VS(LEFT);
            END MERGE;
 EXIT:      IF ERROR THEN RETURN;
          *END*
*PRODUCTION* 1 *CODE*
        /* END OF EXPRESSION  OUTPUT ANSWER */
            IF ANS¬=1 THEN
                OUT=SEP||SEP||SEP||'ALPHA ILLEGAL EXPRESSION';
                ELSE OUT=ARG(VS(LEFT));
            *END*
*PRODUCTION* 3 *CODE*
        /*   ADDITION  */
            CALL FETCH(ARG(VS(RIGHT)),'UNARY');
            IF I(1)=1 THEN CALL BRACKET(VS(RIGHT));
            STEMP='ALPHA +';
            CALL MERGE('1');
            *END*
*PRODUCTION* 4 *CODE*
        /*   SUBTRACTION     */
            CALL FETCH(ARG(VS(RIGHT)),'TYPE UNARY');
            IF I(1)=1 | I(1)=2 | I(2)=1 THEN CALL BRACKET(VS(RIGHT));
            STEMP='ALPHA -';
            CALL MERGE('2');
            *END*
*PRODUCTION* 5 *CODE*
        /*   NEGATION     */
            CALL FETCH(ARG(VS(RIGHT)),'TYPE UNARY');
            IF I(1)=1 | I(1)=2 | I(2)=1 THEN CALL BRACKET(VS(RIGHT));
            PTR=0;
            CALL FETCH(ARG(VS(RIGHT)),'RX LX');
            I(3)=CHAR_SIZE_X-I(2);   /* X TRANSLATION */
            A(1)=DELETE(I(1)+I(3));          /*NEW RX */
            A(2)='1';                        /*NEW UNARY*/
            PTR=0;
            CALL SAVE_ATTR(ARG(VS(RIGHT)),SAVE(FETCH_ATTR(ARG(VS(
                  RIGHT))),'RX UNARY'));
            STEMP='ALPHA -'||CSEP||TRANSLATE(FETCH_BDF(ARG(VS(RIGHT))),
                  I(3),0);
            CALL SAVE_BDF(ARG(VS(RIGHT)),STEMP);
            VS(LEFT)=VS(RIGHT);
            *END*
*PRODUCTION* 8 *CODE*
```

```
        /*     MULTIPLICATION     */
            CALL FETCH(ARG(VS(LEFT)),'TYPE');
            IF I(1)=1 | I(1)=2 THEN CALL BRACKET(VS(LEFT));
            PTR=0;
            CALL FETCH(ARG(VS(RIGHT)),'TYPE UNARY');
            IF I(1)=1 | I(1)=2 | I(2)=1 THEN CALL BRACKET(VS(RIGHT));
            STEMP='ALPHA *';
            CALL MERGE('3');
            *END*
*PRODUCTION* 9 *CODE*
        /*     DIVISION     */
            CALL FETCH(ARG(VS(LEFT)),'TYPE');
            CALL FETCH(ARG(VS(RIGHT)),'TYPE');
            IF I(1)=4 THEN CALL BRACKET(VS(LEFT));
            IF I(2)=4 THEN CALL BRACKET(VS(RIGHT));
            PTR=0;
            CALL FETCH(ARG(VS(LEFT)),'LX RX TY BY');
            CALL FETCH(ARG(VS(RIGHT)),'LX RX TY BY');
            /* CALCULATE TRANSLATION FOR NUMERATOR AND DENOM
               AND COMPOSITE BDF */
            I(11),I(12)=0;
            IF I(2)-I(1) < I(6)-I(5) THEN
                  I(11)= (I(6)-I(5)-(I(2)-I(1)))/2 - I(1);
                  ELSE I(12)=(I(2)-I(1)-(I(6)-I(5)))/2 - I(5);
            STEMP=TRANSLATE(FETCH_BDF(ARG(VS(LEFT))),I(11),
                     CHAR_SIZE_Y/DIVS-I(4));
            STEMP=STEMP||CSEP||'LINE 0 '||DELETE(LUNDER)||
                  ' 0 1 '||DELETE(MAX(I(11)+I(2)+RUNDER,
                  I(12)+I(6)+RUNDER))||' 0';
            STEMP=STEMP||CSEP||TRANSLATE(FETCH_BDF(ARG(VS(RIGHT))),
                      I(12),-CHAR_SIZE_Y/DIVS-I(7));
            CALL SAVE_BDF(ARG(VS(LEFT)),STEMP);
               /*  CALCULATE NEW ATTR */
            A(1)='0';     /*NEW LX */
            A(2)=DELETE(MAX(I(11)+I(2),I(12)+I(6)));    /* NEW RX */
            A(3)=DELETE(I(3)+CHAR_SIZE_Y/DIVS-I(4));   /*NEW TY */
            A(4)=DELETE(I(8)-CHAR_SIZE_Y/DIVS-I(7));   /*NEW BY */
            A(5)='0';    /*NEW MY */
            A(6)='4';   /*NEW TYPE */
            A(7)='0';    /*NEW UNARY*/
            PTR=0;
            CALL SAVE_ATTR(ARG(VS(LEFT)),SAVE(FETCH_ATTR(ARG(VS(LEFT))),
                   'LX RX TY BY MY TYPE UNARY'));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 12 *CODE*
        /*     EXPONENT          */
            CALL FETCH(ARG(VS(LEFT)),'TYPE UNARY');
            IF (I(1)¬=0) | I(2)=1  THEN
                 CALL BRACKET(VS(LEFT));
            PTR=0;
            CALL FETCH(ARG(VS(LEFT)),'RX TY');
            CALL FETCH(ARG(VS(RIGHT)),'LX RX BY TY');
            I(9)=I(1)-I(3);  /* X TRANSLATION */
            I(10)=I(2)-I(5);     /* Y TRANSLATION */
            STEMP=FETCH_BDF(ARG(VS(LEFT)))||CSEP||TRANSLATE(FETCH_BDF(
                   ARG(VS(RIGHT))),I(9),I(10));
            CALL SAVE_BDF(ARG(VS(LEFT)),STEMP);
               /* CALCULATE NEW ATTR */
            A(1)=DELETE(I(9)+I(4));  /*NEW RX */
```

```
            A(2)=DELETE(I(10)+I(6));  /* NEW TY */
            A(3)='5';                   /*NEW TYPE*/
            A(4)='0';                  /* NEW UNARY*/
            PTR=0;
            CALL SAVE_ATTR(ARG(VS(LEFT)),SAVE(FETCH_ATTR(ARG(VS(LEFT))),
                  'RX TY TYPE UNARY'));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 14 *CODE*
        /*   PARENTHESIS    */
            VS(LEFT)=VS(LEFT+1);
            *END*
*PRODUCTION* 15 *CODE*
        /*   WORD  RECOGNITION    */
            IF ANS<5 THEN DO;
                ANS=ANS+1;
                ARG(ANS)=MAKE_PRIMITIVE(VS(LEFT));
                VS(LEFT)=ANS;
                END;
              ELSE DO;
                OUT=SEP||SEP||SEP||'ALPHA ARG OVERFLOW';
                ERROR='1'B;
                GO TO EXIT;
                END;
            *END*
*PRODUCTION* 16 *CODE*
        /*   INTEGER RECOGNITION    */
            IF ANS<5 THEN DO;
                 ANS=ANS+1;
                 ARG(ANS)=MAKE_PRIMITIVE(VS(LEFT));
                 VS(LEFT)=ANS;
                 END;
              ELSE DO;
                OUT=SEP||SEP||SEP||'ALPHA ARG OVERFLOW';
                ERROR='1'B;
                GO TO EXIT;
                END;
            *END*
*END-SEMANTICS*
```

APPENDIX E -- TWO-DIMENSIONAL MATHEMATICAL EXPRESSIONS

MATHEMATICAL SEMANTIC DESCRIPTION

```
//GO.SOURCE DD *
 *SEMANTICS*   SEMANT2   *CODE*
          DCL REAL(20) FLOAT BIN,
              REPLY INT ENTRY(CHAR(*) VAR);
 REPLY:   PROC(S);
          DCL S CHAR(*) VAR;
              DIAG=S;
              ERROR='1'B;
              RETURN;
              END REPLY;
 EXIT:    IF ERROR THEN RETURN;
          ON ERROR BEGIN;
               CALL REPLY('NEGATIVE NUMBER RAISED TO A POWER');
               GO TO EXIT;
               END;
          ON CONVERSION BEGIN;
               CALL REPLY('CONVERSION ERROR');
               GO TO EXIT;
               END;
          ON OVERFLOW BEGIN;
               CALL REPLY('OVERFLOW');
               GO TO EXIT;
               END;
          ON UNDERFLOW BEGIN;
               CALL REPLY('UNDERFLOW');
               GO TO EXIT;
               END;
          ON ZERODIVIDE BEGIN;
               CALL REPLY('ZERODIVIDE');
               GO TO EXIT;
               END;
           *END*
*PRODUCTION* 1 *CODE*
        /*  END OF EXPRESSION   OUTPUT ANSWER  */
            IF ANS¬=1 THEN DO;
                CALL REPLY('ILLEGAL EXPRESSION');
                GO TO EXIT;
                END;
            OUT=REAL(VS(LEFT));
            *END*
*PRODUCTION* 3 *CODE*
        /*    ADDITION    */
            REAL(VS(LEFT))=REAL(VS(LEFT))+REAL(VS(RIGHT));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 4 *CODE*
        /*    SUBTRACTION   */
            REAL(VS(LEFT))=REAL(VS(LEFT))-REAL(VS(RIGHT));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 5 *CODE*
        /*   NEGATION    */
            REAL(VS(RIGHT))=- REAL(VS(RIGHT));
            VS(LEFT)=VS(RIGHT);
            *END*
```

```
*PRODUCTION* 8 *CODE*
        /*   MULTIPLICATION   */
            REAL(VS(LEFT))=REAL(VS(LEFT))*REAL(VS(RIGHT));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 9 *CODE*
        /*     DIVISION      */
            REAL(VS(LEFT))=REAL(VS(LEFT))/REAL(VS(RIGHT));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 12 *CODE*
        /*    EXPONENT    */
            REAL(VS(LEFT))=REAL(VS(LEFT))**REAL(VS(RIGHT));
            ANS=VS(LEFT);
            *END*
*PRODUCTION* 14 *CODE*
        /*   PARENTHESES     */
            VS(LEFT)=VS(LEFT+1);
            *END*
*PRODUCTION* 15 *CODE*
        /*    WORD  RECOGNITION     */
            IF ANS<20 THEN DO;
               ANS=ANS+1;
               IF FETCH_PRIMITIVE(VS(LEFT))='' THEN DO;
                  CALL REPLY('UNDEFINED VARIABLE');
                  RETURN;
                  END;
               DIAG=FETCH_TS(FETCH_PRIMITIVE(VS(LEFT)));
               IF DIAG='' | DIAG=' ' THEN DO;
                  DIAG='';
                  CALL FETCH(FETCH_PRIMITIVE(VS(LEFT)),'VALUE');
                  REAL(ANS)=R(1);
                  VS(LEFT)=ANS;
                  RETURN;
                  END;
                ELSE DO;
                  DIDDLE='1'B;
                  DIAG=LEFT_PAREN||MARKER||DIAG||MARKER||RIGHT_PAREN;
                  ANS=ANS-1;
                  RETURN;
                  END;
               END;
             ELSE DO;
                CALL REPLY('REAL STACK OVERFLOW');
                RETURN;
                END;
            *END*
*PRODUCTION* 16 *CODE*
        /*    INTEGER RECOGNITION    */
            IF ANS<20 THEN DO;
               ANS=ANS+1;
               IF FETCH_PRIMITIVE(VS(LEFT))='' THEN
                     REAL(ANS)=VS(LEFT);
               ELSE DO;
               DIAG=FETCH_TS(FETCH_PRIMITIVE(VS(LEFT)));
               IF DIAG='' | DIAG=' ' THEN DO;
                  DIAG='';
                  CALL FETCH(FETCH_PRIMITIVE(VS(LEFT)),'VALUE');
                  REAL(ANS)=R(1);
                  VS(LEFT)=ANS;
```

```
                RETURN;
                END;
              ELSE DO;
                DIDDLE='1'B;
                DIAG=LEFT_PAREN||MARKER||DIAG||MARKER||RIGHT_PAREN;
                ANS=ANS-1;
                RETURN;
                END;
                 END;
             VS(LEFT)=ANS;
             END;
           ELSE DO;
              CALL REPLY('REAL STACK OVERFLOW');
              RETURN;
              END;
           *END*
*END-SEMANTICS*
```

APPENDIX E -- TWO-DIMENSIONAL MATHEMATICAL EXPRESSIONS

PROCEDURE LIBRARY

```
SCAN:     PROC(A);
          /* DISPLAYS A BY SCANNING BDF- A ASSUMED IN
             PRIMITIVE REPRESENTATION*/
          DCL (ATR,BDF) CHAR (2000) VAR,
              A CHAR(*) VAR,
              LINE BIT(1),
              (ALPHA,BETA,CENX,CENY,INT,RAD) FIXED BIN,
              (B,TYP) CHAR(2000) VAR,
              (X(1000),Y(1000),BL(1000)) FIXED BIN,
              XTR EXT FIXED BIN ,
              YTR EXT FIXED BIN ,
              BLK EXT FIXED BIN ,
              CSEP EXT CHAR(20) VAR,
              BUFADD EXT FIXED BIN,
              I FIXED BIN,
              NEXT INT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                 RETURNS(CHAR(2000) VAR),
              (FETCH_ATTR,FETCH_BDF) EXT ENTRY(CHAR(*) VAR)
                 RETURNS(CHAR(2000) VAR),
              INITIALIZE EXT ENTRY(FIXED BIN),
              FETCH_VALUE EXT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                 RETURNS(CHAR(2000) VAR);
          DCL (PLOTL,PLOTP) EXT ENTRY((*) FIXED BIN,(*) FIXED BIN,
                (*) FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,
                FIXED BIN);
NEXT:     PROC(A,B)  RETURNS(CHAR(2000) VAR);
          /* RETURNS NEXT ELEMENT OF A DELIMITED BY B,
             DELETES FROM A */
          DCL A CHAR(*) VAR,B CHAR(*) VAR,
              TMP CHAR(2000) VAR,I FIXED BIN;
              IF A='' | A=' ' THEN DO;
                 A=''; RETURN('');
                 END;
              I=INDEX(A,B);
              IF I=0 THEN DO;
                 TMP=A;  A='';
                 END;
              ELSE DO;
                 TMP=SUBSTR(A,1,I-1);
                 A=SUBSTR(A,I+LENGTH(B));
                 END;
              RETURN(TMP);
              END NEXT;
ARC_GEN:   PROC(I,INT,CENX,CENY,RAD,ALPHA,BETA);
           /* GENERATES VECTOR REPRESENTATION FOR ARC */
           DCL (I,INT,CENX,CENY,RAD,ALPHA,BETA) FIXED BIN,
               (J,N) FIXED BIN,
               DELTA FLOAT BIN;
               N=ABS(BETA-ALPHA)/5 +1;
               IF N=1 THEN N=2;
               DELTA=(BETA-ALPHA)/(N-1);
               DO J=1 TO N;
                  IF I<1000 THEN I=I+1;
                  IF J=1 THEN BL(I)=0; ELSE BL(I)=INT;
                  X(I)=CENX+RAD*COSD(ALPHA+(J-1)*DELTA)+XTR;
```

```
   Y(I)=CENY+RAD*SIND(ALPHA+(J-1)*DELTA)+YTR;
   END;
END ARC_GEN;
   I=0;      LINE='1'B;
   ATR=FETCH_ATTR(A);
   BDF=FETCH_BDF(A);
   DO WHILE (BDF¬='');
      B=NEXT(BDF,CSEP);
      TYP=NEXT(B,' ');
      IF TYP='LINE' THEN DO;
         IF ¬ LINE THEN DO;
             CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
             I=0;  LINE='1'B;
             END;
         DO WHILE(B¬='');
            IF I<1000  THEN I=I+1;
            BL(I)=BLK*NEXT(B,' ');
            X(I)=XTR+NEXT(B,' ');
            Y(I)=YTR+NEXT(B,' ');
            END;
         END;
      IF TYP='POINT' THEN DO;
         IF  LINE THEN DO;
             CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
             I=0;  LINE='0'B;
             END;
         DO WHILE(B¬='');
            IF I<1000  THEN I=I+1;
            BL(I)=BLK*NEXT(B,' ');
            X(I)=XTR+NEXT(B,' ');
            Y(I)=YTR+NEXT(B,' ');
            END;
         END;
      ELSE IF TYP='ALPHA' THEN DO;
         IF I<1000 THEN I=I+1;
         BL(I)=0;
         X(I)=XTR;
         Y(I)=YTR;
         IF LINE THEN CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
            ELSE CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
         I=0;   LINE='1'B;
         CALL STRLAR(B,BUFADD);
         END;
      ELSE IF TYP='ARC' THEN DO;
         IF ¬ LINE THEN DO;
             CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
             I=0;  LINE='1'B;
             END;
           INT= BLK*NEXT(B,' ');
           CENX=NEXT(B,' ');
           CENY=NEXT(B,' ');
           RAD=NEXT(B,' ');
           ALPHA=NEXT(B,' ');
           BETA=NEXT(B,' ');
           CALL ARC_GEN(I,INT,CENX,CENY,RAD,ALPHA,BETA);
           END;
      ELSE IF TYP='TRANS' THEN DO;
           XTR XTR+NEXT(B,' ');
           YTR=YTR+NEXT(B,' ');
           END;
```

```
ELSE IF TYP='UNTRAN' THEN DO;
     XTR=XTR-NEXT(B,' ');
     YTR=YTR-NEXT(B,' ');
     END;
ELSE IF TYP='BLANK' THEN BLK=0;
ELSE IF TYP='UNBLANK' THEN BLK=1;
END;
IF I>0 THEN DO;
  IF LINE THEN CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
    ELSE CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
  END;
CALL LOWER(BUFADD);
END SCAN;
```

```
SCAN1:      PROC(A);
          /*CONVERTS BDF TO CHARACTER ARRAY EXTERNAL */
          DCL A CHAR(*) VAR,CH(-50:50) EXT CHAR(80),
              (B,TYP,BDF) CHAR(2000) VAR,
              (XTR,YTR) EXT FIXED BIN,
              CSEP EXT CHAR(20) VAR,
              (I,I1,I2,B1,B2,J1,J2) FIXED BIN,
              NEXT INT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                    RETURNS(CHAR(2000) VAR),
              FETCH_BDF EXT ENTRY(CHAR(*) VAR)
                    RETURNS (CHAR(2000) VAR),
              SAVE INT ENTRY(FIXED BIN) RETURNS(BIT(1));
SAVE:     PROC(I)  RETURNS(BIT(1));
          /* TRUE IF CH(I) CONTAINS NON BLANK OR NON | */
          DCL (I,J) FIXED BIN;
             IF CH(I)=' ' | CH(I)='' THEN RETURN('0'B);
             DO J=1 TO 80;
                IF SUBSTR(CH(I),J,1)¬=' ' & SUBSTR(CH(I),J,1)¬='|'
                   THEN RETURN('1'B);
                END;
             RETURN('0'B);
             END SAVE;
NEXT:     PROC(A,B)  RETURNS(CHAR(2000) VAR);
          /*RETURNS NEXT ELEMENT OF A DELIMINATED BY B,
            DELETES FROM A */
          DCL (A,B) CHAR(*) VAR,TMP CHAR(2000) VAR,I FIXED BIN;
             IF A='' | A=' ' THEN DO;
                A=''; RETURN('');
                END;
             I=INDEX(A,B);
             IF I=0 THEN DO;
                TMP=A; A='';
                END;
                ELSE DO;
                   TMP=SUBSTR(A,1,I-1);
                   A=SUBSTR(A,I+LENGTH(B));
                   END;
             RETURN(TMP);
             END NEXT;
              ON SUBSCRIPTRANGE BEGIN;
                   CH(50)='CH(*) SUBSCRIPT RANGE VIOLATION';
                   GO TO EXIT;
                   END;
              ON STRINGRANGE BEGIN;
                   CH(50)='CH(*) STRING RANGE VIOLATION';
                   GO TO EXIT;
                   END;
              DO I=-50 TO 50;
                   CH(I)=(80)' ';
                   END;
              BDF=FETCH_BDF(A);
              DO WHILE(BDF ¬= '');
                  B=NEXT(BDF,CSEP);
                  TYP=NEXT(B,' ');
                  IF TYP='TRANS' THEN DO;
                     XTR=XTR+NEXT(B,' ');
                     YTR=YTR+NEXT(B,' ');
                     END;
                  ELSE IF TYP='UNTRAN' THEN DO;
                     XTR=XTR-NEXT(B,' ');
```

```
                    YTR=YTR-NEXT(B,' ');
                    END;
                 ELSE IF TYP='ALPHA' THEN
(SUBSCRIPTRANGE,STRINGRANGE):
                        SUBSTR(CH(YTR),XTR,LENGTH(B))=B;
                 ELSE IF TYP='LINE' THEN DO;
                    B2=NEXT(B,' ');
                    I2=NEXT(B,' ');
                    J2=NEXT(B,' ');
                    DO WHILE(B¬='');
                       B1=B2;
                       I1=I2;
                       J1=J2;
                       B2=NEXT(B,' ');
                       I2=NEXT(B,' ');
                       J2=NEXT(B,' ');
                       IF B2¬=0 THEN DO;
                          /* IS IT VERTICAL LINE */
                          IF J2¬=J1 THEN DO I=MIN(J1,J2)+1 TO
                                   MAX(J1,J2)-1 ;
(SUBSCRIPTRANGE,STRINGRANGE):
                                SUBSTR(CH(I+YTR),XTR+I1,1)='|';
                                END;
                          /* OTHERWISE IT IS HORIZONTAL LINE */
                          ELSE DO I=MIN(I1,I2) TO MAX(I1,I2);
(SUBSCRIPTRANGE,STRINGRANGE):
                                SUBSTR(CH(J1+YTR),XTR+I,1)='-';
                                END;
                          END;
                       END;
                    END;
                 END;
EXIT:            I1=50;
                 DO I=50 TO -50 BY -1;
                    IF SAVE(I) THEN DO;
                       CH(I1)=CH(I);
                       I1=I1-1;
                       END;
                    END;
                 IF I1<-50 THEN I1=-50;
                 CH(-50)=I1;
                    END SCAN1;


PRINTER:   PROC;
           /*PRINTS NON BLANK CH */
           DCL CH(-50:50) EXT CHAR(80),I FIXED BIN;
                ON CONVERSION ONSOURCE='49';
              PUT LIST(' ') SKIP(4);
              DO I=50 TO CH(-50)+1 BY -1;
                 PUT EDIT(CH(I))(COL(21),A);
                 END;
EXIT:
            END PRINTER,
```

```
PRINTER:    PROC;
            /* TYPES NON BLACK CH */
            DCL CH(-50:50) EXT CHAR(80), I FIXED BIN;
                ON CONVERSION ONSOURCE='49';
                DO I=1 TO 4;
                   CALL OUTPUT(' ');
                   END;
                DO I=50 TO CH(-50)+1 BY -1;
                   CALL OUTPUT(CH(I));
                   END;
                DO I=1 TO 4;
                   CALL OUTPUT(' ');
                   END;
            END PRINTER;


MAKE_PRIMITIVE:   PROC(S)   RETURNS(CHAR(2000) VAR);
        /* MAKES THE TV PART OF A PRIMITIVE */
        DCL (ASEP,SEP,CONNECTOR) EXT CHAR(20) VAR,
           S CHAR(*) VAR, T CHAR(2000) VAR,
           (A(5),B(5)) CHAR(100) VAR,
           (CHAR_SIZE_X,CHAR_SIZE_Y,CDIV,NSPACE) EXT FIXED BIN,
           I FIXED BIN,
           DELETE INT ENTRY (FIXED BIN) RETURNS(CHAR(100) VAR);
DELETE:  PROC(I)  RETURNS(CHAR(100) VAR);
         DCL (I,J) FIXED BIN, S CHAR(100) VAR;
            S=I;
            DO WHILE (SUBSTR(S,1,1)=' ');
               S=SUBSTR(S,2);
               END;
            RETURN(S);
            END DELETE;
            I=LENGTH(S);
            B(1)='LX';  B(2)='RX'; B(3)='BY'; B(4)='MY'; B(5)='TY';
            A(1)='0';
            A(2)=DELETE(I*(CHAR_SIZE_X/CDIV)+NSPACE);
            A(3)=DELETE(-(CHAR_SIZE_Y+1)/2);
            A(4)='0';
            A(5)=DELETE((CHAR_SIZE_Y+1)/2);
            T=SEP||SEP;
            DO I=1 TO 5;
               T=T||B(I)||CONNECTOR||A(I)||ASEP;
               END;
            T=T||SEP||'ALPHA '||S;
            RETURN(T);
            END MAKE_PRIMITIVE;
```

```
SETUPI:   PROC;
         /* SETS UP SCOPE */
         DCL NBYTE FIXED BIN(31,0) INITIAL(5000),
             (BUFADD,BUFPTRS(8)) EXT FIXED BIN,
             UNIT FIXED BIN(31,0) INITIAL(1);
             DO BUFADD=1 TO 8; BUFPTRS(BUFADD)=0; END;
             CALL GASBUF(UNIT,NBYTE);
             CALL GINIT(BUFADD);
             RETURN;
SETUPO:      ENTRY;     RETURN;
             END SETUPI;


OUTPUT:     PROC(C);
            /* OUTPUTS C  */
            DCL A CHAR(2000) VAR, B CHAR(72) VAR,I FIXED BIN INITIAL(1),
                LEN FIXED BIN INITIAL(45),C CHAR(*) VAR,
                BUFADD EXT FIXED BIN,
                NEXT INTERNAL ENTRY(CHAR(2000) VAR) RETURNS
                    (CHAR(72) VAR);
NEXT:       PROC(A)  RETURNS(CHAR(72) VAR);
            /*RETURNS NEXT LINE, LENGTH<=LEN */
            DCL A CHAR(2000) VAR,B CHAR(72) VAR, I FIXED BIN;
              IF LENGTH(A)<=LEN THEN DO;
                 B=A;  A='';
                 RETURN(B);
                 END;
              IF INDEX(A,' ')>LEN | INDEX(A,' ')=0 THEN DO;
                 B=SUBSTR(A,1,LEN);
                 A=SUBSTR(A,LEN+1);
                 RETURN(B);
                 END;
              DO I=LEN TO 1 BY -1;
                 IF SUBSTR(A,I,1)=' ' THEN DO;
                    B=SUBSTR(A,1,I);
                    A=SUBSTR(A,I+1);
                    RETURN(B);
                    END;
                 END;
              END NEXT;
              A=C;
              DO WHILE(A¬='');
                 CALL DNSPAC(I,BUFADD);
                 B=NEXT(A);
                 CALL CONTMO(B,BUFADD);
                 END;
               END OUTPUT;


INPUT:      PROC(A);
            /* GETS INPUT TO A */
            DCL A CHAR(*) VAR,B CHAR(72) VAR, C BIT(1),
               BUFADD EXT FIXED BIN;
               CALL GETDAT(B,BUFADD,C);
               A=B;
               END INPUT;
```

```
LOWER:    PROC(BUFADD);
          /* LEAVES BEAM AT LOWER LEFT USING BUFADD FOR POINTER*/
          DCL B(0:2) BIT(32),
              (L,M,YY,XXX) FIXED BIN(31,0),
              BUFADD  FIXED BIN;
              CALL CONV('2A802A02',B(0));
              CALL CONV('40000200',B(1));
              CALL CONV('2A802A45',B(2));
              L=BUFADD;  M=12;  BUFADD=BUFADD+12; YY=0;
              XXX=BUFADD;
              CALL GTRAN(XXX);
              CALL GWTSTR(L,M,B(0),YY);
              END LOWER;


PLOTL:    PROC(INT,X,Y,N,D,DEC,LOC);
          /*PLOTS THE N VECTORS IN S,Y USING LOC AS THE BUFFER PTR,
            IF INT>=DEC THEN ON ELSE OFF */
          DCL INT(D) FIXED BIN, /*INTENSITY LEVEL*/
              X(D) FIXED BIN,  /*X-COORDINATES */
              Y(D) FIXED BIN,  /*Y-COORDINATES */
              N FIXED BIN ,  /*NUM PTS TO PLOT */
              D FIXED BIN,  /*DIMEN OF X AND Y */
              DEC FIXED BIN, /*DECISION LEVEL */
              LOC FIXED BIN, /*BUFFER PTR */
              XX BIT(16),
              (I,K) FIXED BIN,
              (L,M,YY,XXX) FIXED BIN(31,0),
              B(0:N) BIT(32);
              CALL CONV('2A802A02',B(0));  /*VECTOR MODE*/
              GO TO START;
PLOTP:    ENTRY(INT,X,Y,N,D,DEC,LOC);
          /* PLOTS POINTS  */
              CALL CONV('2A802A00',B(0));   /*POINT MODE */
START:        DO I=1 TO N;
                 B(I)='01'B;
                 IF INT(I)>=DEC THEN SUBSTR(B(I),2,1)='0'B;
                 XX=UNSPEC(X(I));
                 SUBSTR(B(I),5,10)=SUBSTR(XX,7);
                 XX=UNSPEC(Y(I));
                 SUBSTR(B(I),21,10)=SUBSTR(XX,7);
                 END;
              L=LOC;  M=4*(N+1);  LOC=LOC+M;
              YY=0;  XXX=LOC;
              CALL GTRAN(XXX);
              CALL GWTSTR(L,M,B(0),YY);
              END PLOTL;
```

```
SLAVX:    PROC OPTIONS(MAIN);
          DCL A CHAR(80) VAR,B CHAR(2000) VAR,C CHAR(20) VAR;
          DCL BUFPTRS(8) EXT FIXED BIN;
          DO I=1 TO 8; BUFPTRS(I)=0;  END;
          OPEN FILE(XYZ) PRINT;
          ON ENDFILE GO TO EXIT;
          GET LIST(C);
          PUT FILE(XYZ) EDIT('C=',C)(SKIP,2 A);
          DO WHILE ('1'B);
              B=''; A='';
              DO WHILE(INDEX(A,C)=0);
                 B=B||A;
                 GET EDIT(A)(SKIP,A(80));
                 PUT FILE(XYZ) EDIT(A)(SKIP,A);
                 END;
              B=B||SUBSTR(A,1,INDEX(A,C)-1);
              CALL SLAVES(B);
              END;
EXIT:     PUT FILE(XYZ)  EDIT('NORMAL TERMINATION')(SKIP,A);
          END SLAVX;


SETUPI:   PROC;
          DCL MILTIE EXT ENTRY(FIXED BIN(15),CHAR(*) VAR,CHAR(*) VAR,
              CHAR(*) VAR, FIXED BIN(15)),B FIXED BIN(15),
              (C,D) CHAR(3) VAR;
          DCL (BUFADD,BUFPTRS(8)) EXT FIXED BIN;
              DO BUFADD=1 TO 8;  BUFPTRS(BUFADD)=0; END;
              CALL MILTIE(13,'GEMS',C,D,B);
              CALL MILTIE(22,C,C,D,B);
              IF D ¬= 'JEG' THEN DO;
                 CALL MILTIE(2,'USER='||D,C,C,B);
                 CALL MILTIE(6,'',C,D,B);
                 END;
              RETURN;
SETUPO:    ENTRY;
          END SETUPI;
```

```
INPUT:   PROC(A);
         /*TYPEWRITER INPUT/OUTPUT */
         DCL MILTIE EXT ENTRY(FIXED BIN(15),CHAR(*) VAR,
             CHAR(*) VAR,CHAR(*) VAR,FIXED BIN(15)),
             A CHAR(*) VAR,B FIXED BIN(15), (C,D) CHAR(1) VAR;
             CALL MILTIE(1,' ',' ',A,B);
             IF A='*RESTART*' THEN CALL MILTIE(6,'','','',B);
             RETURN;
OUTPUT: ENTRY(A);
         DCL E CHAR(1),AA CHAR(2000) VAR,BB CHAR(60) VAR,
             NEXT INT ENTRY(CHAR(*) VAR) RETURNS(CHAR(60) VAR);
NEXT:    PROC(A) RETURNS(CHAR(60) VAR);
         /* RETURNS NEXT LINE */
         DCL A CHAR(*) VAR,B CHAR(60) VAR,I FIXED BIN;
            IF LENGTH(A) <= 60 THEN DO;
                B=A;  A='';  RETURN(B);
                END;
            IF INDEX(A,' ')>60 | INDEX(A,' ') = 0 THEN DO;
                B=SUBSTR(A,1,60);  A=SUBSTR(A,61); RETURN(B);
                END;
            DO I=60 TO 1 BY -1;
                IF SUBSTR(A,I,1)=' ' THEN DO;
                   B=SUBSTR(A,1,I);  A=SUBSTR(A,I+1);
                   RETURN(B);
                   END;
                END;
            END NEXT;
            UNSPEC(E)='00010101'B;
            AA=A;
            DO WHILE (AA ¬= '');
               BB=NEXT(AA);
               IF BB='' | BB=' ' THEN BB=' ';
                  ELSE DO WHILE(SUBSTR(BB,LENGTH(BB),1)=' ');
                        BB=SUBSTR(BB,1,LENGTH(BB)-1);
                        END;
               CALL MILTIE(2,BB,C,D,B);
               CALL MILTIE(2,E,C,D,B);
               END;
            END INPUT;
```

APPENDIX G -- A DRAWING SYSTEM

CONTROL DESCRIPTION

```
//GO.SOURCE DD *
 *CONTROL* 'SELECT OPERATOR OR FUNCTION'
 MAX_PRIM *=* 10 TERMINATOR *=* 'NO' RIGHT_PAREN *=* ')'
 LEFT_PAREN *=* '(' CSEP *=* ':' CONNECTOR *=* '='
 QUOTES *=* '"' SEP *=* ';'  MARKER *=* '' ASEP *=* ' '
 LENGTH_PRIM *=* 200
 ¬  #
          *CODE*
        DCL SCAN1 EXT ENTRY(CHAR(*) VAR,CHAR(1),CHAR(1),
                  FIXED BIN,FIXED BIN);
        DCL SEMANT EXT ENTRY;
        DCL DBG EXT BIT(1) INITIAL('0'B);
        *END*
*FUNCTION*  'FUNCTIONS'
       *NAME* *=* DISPLAY 'SELECT ARGUMENT'
          0 PRIMITIVE TYPE_A */* 0 RESULT TYPE_A
          *CODE*
             ARG(2)=FETCH_PRIMITIVE(ARG(1));
             IF ARG(2)='' THEN ARG(2)=ARG(1);
             CALL INITIALIZE(5);
             CALL OUTPUTT(ARG(2));
             *END*
       *NAME* *=* ASSIGN 'SELECT TARGET'
          0 PRIMITIVE TYPE_A
          *CODE*
             ARG(2)=RES_STACK(10);
             ARG(1)=FETCH_PRIMITIVE(ARG(1));
             CALL SAVE_TS(ARG(1),ARG(2));
             ARG(2)='';
             CALL SAVE_ATTR(ARG(1),ARG(2));
             CALL SAVE_BDF(ARG(1),ARG(2));
             CALL SAVE_PRIMITIVE(ARG(1));
             *END*
       *NAME* *=* POP *NONE*
          *CODE*
             ARG(1)=POP;
             CALL DISPLAY_RESULT('RESULT = ');
             *END*
       *NAME* *=* DEFINE *NONE*
             *CODE*
                CALL DEFINE;
                CALL DISPLAY_PRIMITIVES('PRIMITIVES');
                CALL DISPLAY_RESULT('RESULT = ');
                *END*
       *NAME* *=* DRAW *NONE*
          *CODE*
             ARG(1)=RES_STACK(10);
             ARG(2)='';
             CALL PARSER(ARG(1),ARG(1),ARG(2),SEMANT);
             IF DBG THEN DO;
                 CALL INITIALIZE(4);
                 CALL OUTPUTT(ARG(1));
                 END;
             CALL INITIALIZE(5);
             CALL SCAN(ARG(1));
```

```
            *END*
        *NAME* *=* PRINT *NONE*
           *CODE*
              ARG(1)=RES_STACK(10);
              ARG(2)='';
              CALL PARSER(ARG(1),ARG(1),ARG(2),SEMANT);
              PUT EDIT(RES_STACK(10))(PAGE,COL(20),A);
              IF DBG THEN DO;
                   PUT EDIT(ARG(1))(SKIP(4),A);
                   END;
              CALL SCAN1(ARG(1),' ','X',60,50);
              *END*
        *NAME* *=* DBG *NONE*
           *CODE*
              DBG=¬DBG;
              *END*
        *NAME* *=* TERM *NONE*
           *CODE*
              GO TO EXIT2;
              *END*
             *END*
*CONSTRUCTION* 'CONSTRUCTIONS'
        *NAME* *=* + 'SELECT FIRST ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           'SELECT SECOND ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           *CODE*
              ARG(1)=ARG(1)||MARKER||'+'||MARKER||ARG(2);
              CALL UPDATE(ARG(1));
              CALL DISPLAY_RESULT('RESULT = ');
              *END*
        *NAME* *=* - 'SELECT FIRST ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           'SELECT SECOND ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           *CODE*
              ARG(1)=ARG(1)||MARKER||'-'||MARKER||ARG(2);
              CALL UPDATE(ARG(1));
              CALL DISPLAY_RESULT('RESULT = ');
              *END*
        *NAME* *=* & 'SELECT FIRST ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           'SELECT SECOND ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           *CODE*
              ARG(1)=ARG(1)||MARKER||'&'||MARKER||ARG(2);
              CALL UPDATE(ARG(1));
              CALL DISPLAY_RESULT('RESULT = ');
              *END*
        *NAME* *=* * 'SELECT FIRST ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           'SELECT SECOND ARGUMENT'
           1 PRIMITIVE TYPE_A */* 1 RESULT TYPE_B
           *CODE*
              ARG(1)=ARG(1)||MARKER||'*'||MARKER||ARG(2);
              CALL UPDATE(ARG(1));
              CALL DISPLAY_RESULT('RESULT = ');
              *END*
        *NAME* *=* ¬ 'SELECT ARGUMENT'
           0 PRIMITIVE TYPE_A */* 0 RESULT TYPE_B
```

```
      *CODE*
         ARG(1)='¬'||MARKER||ARG(1);
         CALL UPDATE(ARG(1));
         CALL DISPLAY_RESULT('RESULT = ');
         *END*
   *NAME* *=* # 'SELECT ARGUMENT'
      0 PRIMITIVE TYPE_A */* 0 RESULT TYPE_B
      *CODE*
         ARG(1)='#'||MARKER||ARG(1);
         CALL UPDATE(ARG(1));
         CALL DISPLAY_RESULT('RESULT = ');
         *END*
     *END*
*END-CONTROL*
```

APPENDIX G -- A DRAWING SYSTEM

DEFINITION DESCRIPTION

```
//GO.SOURCE DD *
*DEFINITION* DEFINE
 *NAME* 'ENTER  PRIMITIVE  NAME'
*TS*  'ENTER TS'
*ATTR-LIST* 'ENTER ATTRIBUTES'
 TAILX=NUMBER
TAILY=NUMBER
HEADX=NUMBER
HEADY=NUMBER
XMIN=NUMBER
XMAX=NUMBER
YMIN=NUMBER
YMAX=NUMBER
*BDF-FILE* 'SELECT TYPE CONSTRUCTION'
 LINE # 'INTENSITY' NUMBER 'X-COORD' NUMBER 'Y-COORD' NUMBER *END*
 POINT # 'INTENSITY' NUMBER 'X-COORD' NUMBER 'Y-COORD' NUMBER *END*
 ALPHA 1 'TEXT' TEXT *END*
 ARC 1 'INTENSITY' NUMBER 'CENTER-X' NUMBER 'CENTER-Y' NUMBER
       'RADIUS' NUMBER 'INITIAL ANGLE' NUMBER 'FINAL ANGLE' NUMBER
*END-DEFINITION*
```

APPENDIX G -- A DRAWING SYSTEM

GRAPHIC SYNTAX DESCRIPTION

```
//GO.SYNDATA DD *
 SEQUENCE='PICTURE' PARSER_NAME='PARSER' QUOTES=''''
 TERMINAL='END'    ;
/*
//GO.SYNTAX DD *
*SYNTAX*
PICTURE *::=* EXPR *;*
EXPR *::=* EXPR- *NO-SEMANT* *;*
EXPR- *::=* EXPR- + TERM *;*
 *::=* EXPR- - TERM *;*
 *::=* EXPR- & TERM *;*
 *::=* EXPR- * TERM *;*
 *::=* TERM *NO-SEMANT* *;*
TERM *::=* FACTOR *NO-SEMANT* *;*
 *::=* ¬ TERM *;*
 *::=* # TERM *;*
FACTOR *::=* WORD *;*
 *::=* INTEGER *;*
 *::=* ( EXPR )
*END-SYNTAX*
```

APPENDIX G -- A DRAWING SYSTEM

GRAPHIC SEMANTIC DESCRIPTION

```
//GO.SOURCE DD *
*SEMANTICS* SEMANT *CODE*
    DCL ARG(5) EXT CHAR(2000) VAR,
        (XTRAN,YTRAN) FIXED BINARY,
        TRAN INT ENTRY(FIXED BIN,FIXED BIN, FIXED BIN,FIXED BIN);
          IF ERROR THEN RETURN;
 TRAN:  PROC(II,J,X,Y);
        /*  TRANSLATES ARG(J) TO X,Y MERGES ARG(II) AND ARG(J) NEW
            HEAD IS HEAD OF ARG(J)+TRANSLATION--HX AND HY ASSUMED
            IN I(1),I(2)..CALCULATES NEW XY MIN MAX SETS ANS=II */
        DCL (II,J,X,Y) FIXED BIN;
            PTR=6;
            CALL FETCH(ARG(II),'XMIN XMAX YMIN YMAX');
            CALL FETCH(ARG(J),'XMIN XMAX YMIN YMAX');
            A(1)=DELETE(I(1)+X);          /*NEW HEADX */
            A(2)=DELETE(I(2)+Y);          /*NEW HEADY*/
            A(3)=DELETE(MIN(I(7),I(11)+X));     /*NEW XMIN*/
            A(4)=DELETE(MAX(I(8),I(12)+X));     /*NEW XMAX*/
            A(5)=DELETE(MIN(I(9),I(13)+Y));     /*NEW YMIN*/
            A(6)=DELETE(MAX(I(10),I(14)+Y));    /*NEW YMAX*/
            A(7)=' '||DELETE(X);                /*X TRANS */
            A(8)=' '||DELETE(Y);                /*Y TRANS*/
            PTR=0;
            CALL SAVE_ATTR(ARG(II),SAVE(FETCH_ATTR(ARG(II)),
               'HEADX HEADY XMIN XMAX YMIN YMAX'));
            ARG(J)=FETCH_BDF(ARG(II))||CSEP||'TRANS'||A(7)||A(8)||CSEP||
                   FETCH_BDF(ARG(J))||CSEP||'UNTRAN'||A(7)||A(8);
            CALL SAVE_BDF(ARG(II),ARG(J));
            ANS=II;
            RETURN;
            END TRAN;
 RECOG:   PROC;
         /* FETCHES PRIMITIVE TO ARG IF ROOM AND SETS XY MIN MAX
            IF REQUIRED FETCH TS */
            IF ANS<5 THEN DO;
               ANS=ANS+1;
               ARG(ANS)=FETCH_PRIMITIVE(VS(LEFT));
               VS(LEFT)=ANS;
               DIAG=FETCH_TS(ARG(ANS));
               IF DIAG='' | DIAG=' ' THEN RETURN;
                ELSE DO;
                   /*RETURN TS IN PROPER FORM */
                   DIDDLE='1'B;
                   DIAG=LEFT_PAREN||MARKER||DIAG||MARKER||RIGHT_PAREN;
                   ANS=ANS-1;
                   RETURN;
                   END;
               END;
             ELSE DO;
               OUT=SEP||SEP||SEP||'ALPHA ARG OVERFLOW';
               ERROR='1'B;
               END;
            END RECOG;
         *END*
*PRODUCTION* 1 *CODE*
```

```
        /* OUTPUT ANSWER */
           IF ANS¬=1 THEN OUT=SEP||SEP||SEP||'ALPHA ILLEGAL EXPRESSION';
             ELSE OUT=ARG(VS(LEFT));
           *END*
*PRODUCTION* 3 *CODE*
        /* + OPERATOR */
           CALL FETCH(ARG(VS(RIGHT)),'HEADX HEADY TAILX TAILY');
           CALL FETCH(ARG(VS(LEFT)),'HEADX HEADY');
           XTRAN=I(5)-I(3);
           YTRAN=I(6)-I(4);
           CALL TRAN(VS(LEFT),VS(RIGHT),XTRAN,YTRAN);
           *END*
*PRODUCTION* 4 *CODE*
        /* - OPERATOR*/
           CALL FETCH(ARG(VS(RIGHT)),'HEADX HEADY');
           CALL FETCH(ARG(VS(LEFT)),'HEADX HEADY');
           XTRAN=I(3)-I(1);
           YTRAN=I(4)-I(2);
           CALL TRAN(VS(LEFT),VS(RIGHT),XTRAN,YTRAN);
           *END*
*PRODUCTION* 5 *CODE*
        /*  & OPERATOR */
           CALL FETCH(ARG(VS(RIGHT)),'HEADX HEADY TAILX TAILY');
           CALL FETCH(ARG(VS(LEFT)),'TAILX TAILY');
           XTRAN=I(5)-I(3);
           YTRAN=I(6)-I(4);
           CALL TRAN(VS(LEFT),VS(RIGHT),XTRAN,YTRAN);
           *END*
*PRODUCTION* 6 *CODE*
        /*   * OPERATOR */
           CALL FETCH(ARG(VS(RIGHT)),'HEADX HEADY TAILX TAILY');
           CALL FETCH(ARG(VS(LEFT)),'HEADX HEADY TAILX TAILY');
           XTRAN=I(7)-I(3);  I(9)=I(1)+XTRAN;
           YTRAN=I(8)-I(4);  I(10)=I(2)+YTRAN;
           IF I(9)¬=I(5) | I(10)¬=I(6) THEN DO;
                OUT=SEP||SEP||SEP||'ALPHA * ERROR';
                ERROR='1'B;
                RETURN;
                END;
            ELSE CALL TRAN(VS(LEFT),VS(RIGHT),XTRAN,YTRAN);
           *END*
*PRODUCTION* 9 *CODE*
        /* ¬ OPERATOR */
           CALL SAVE_BDF(ARG(VS(RIGHT)),'BLANK'||CSEP||
               FETCH_BDF(ARG(VS(RIGHT)))||CSEP||'UNBLANK');
           VS(LEFT)=VS(RIGHT);
           *END*
*PRODUCTION* 10 *CODE*
        /* # OPERATOR */
           CALL FETCH(ARG(VS(RIGHT)),'TAILX TAILY HEADX HEADY');
           PTR=0;
           CALL SAVE_ATTR(ARG(VS(RIGHT)),SAVE(FETCH_ATTR(ARG(VS(RIGHT)))
               ,'HEADX HEADY TAILX TAILY'));
           VS(LEFT)=VS(RIGHT);
           *END*
*PRODUCTION* 11 *CODE*
        /* WORD RECOGNITION */
           CALL RECOG;
           *END*
*PRODUCTION* 12 *CODE*
        /* INTEGER RECOGNITION */
           CALL RECOG;
           *END*
*PRODUCTION* 13 *CODE*
        /*  () RECOG */
           VS(LEFT)=VS(LEFT+1);
           *END*
*END-SEMANTICS*
```

APPENDIX G -- A DRAWING SYSTEM

PROCEDURE LIBRARY

```
SCAN:     PROC(A);
          /* DISPLAYS A ON SCOPE BY SCANNING BDF- A ON SCOPE ASSUMED IN
             PRIMITIVE REPRESENTATION*/
          DCL (ATR,BDF) CHAR (2000) VAR,
              A CHAR(*) VAR,
              LINE BIT(1),
              (ALPHA,BETA,CENX,CENY,INT,RAD) FIXED BIN,
              (B,TYP) CHAR(2000) VAR,
              (X(1000),Y(1000),BL(1000)) FIXED BIN,
              XTR  FIXED BIN ,
              YTR  FIXED BIN ,
              BLK  FIXED BIN ,
              DBG EXT BIT(1) INITIAL('0'B),
              (DIV,XMIN,XMAX,YMIN,YMAX) FIXED BIN,
              CSEP EXT CHAR(20) VAR,
              BUFADD EXT FIXED BIN,
              I FIXED BIN,
              NEXT INT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                 RETURNS(CHAR(2000) VAR),
              (FETCH_ATTR,FETCH_BDF) EXT ENTRY(CHAR(*) VAR)
                 RETURNS(CHAR(2000) VAR),
              INITIALIZE EXT ENTRY(FIXED BIN),
              FETCH_VALUE EXT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                 RETURNS(CHAR(2000) VAR);
          DCL (PLOTL,PLOTP) EXT ENTRY((*) FIXED BIN,(*) FIXED BIN,
                (*) FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN,
                FIXED BIN),
               ARC_GEN INT ENTRY(FIXED BIN,FIXED BIN,FIXED BIN,
                         FIXED BIN,FIXED BIN,FIXED BIN,FIXED BIN);
NEXT:     PROC(A,B)  RETURNS(CHAR(2000) VAR);
          /* RETURNS NEXT ELEMENT OF A DELIMITED BY B,
             DELETES FROM A */
          DCL A CHAR(*) VAR,B CHAR(*) VAR,
              TMP CHAR(2000) VAR,I FIXED BIN;
              IF A='' | A=' ' THEN DO;
                 A=''; RETURN('');
                 END;
              DO WHILE(SUBSTR(A,1,1)=' ');
                A=SUBSTR(A,2);
                END;
              I=INDEX(A,B);
              IF I=0 THEN DO;
                 TMP=A;  A='';
                 END;
              ELSE DO;
                 TMP=SUBSTR(A,1,I-1);
                 A=SUBSTR(A,I+LENGTH(B));
                 END;
              RETURN(TMP);
              END NEXT;
ARC_GEN:   PROC(I,INT,CENX,CENY,RAD,ALPHA,BETA);
           /* GENERATES VECTOR REPRESENTATION FOR ARC */
           DCL (I,INT,CENX,CENY,RAD,ALPHA,BETA) FIXED BIN,
                (J,N) FIXED BIN,
                DELTA FLOAT BIN;
```

```
N=ABS(BETA-ALPHA)/5 +1;
IF N=1 THEN N=2;
DELTA=(BETA-ALPHA)/(N-1);
DO J=1 TO N;
   IF I<1000 THEN I=I+1;
   IF J=1 THEN BL(I)=0; ELSE BL(I)=INT;
   X(I)=(CENX+RAD*COSD(ALPHA+(J-1)*DELTA))/DIV;
   X(I)=X(I)+XTR;
   Y(I)=(CENY+RAD*SIND(ALPHA+(J-1)*DELTA))/DIV;
   Y(I)=Y(I)+YTR;
   END;
END ARC_GEN;
   I=0;      LINE='1'B;       BLK=1;
   ATR=FETCH_ATTR(A);
   XMIN=FETCH_VALUE(ATR,'XMIN');
   XMAX=FETCH_VALUE(ATR,'XMAX');
   YMIN=FETCH_VALUE(ATR,'YMIN');
   YMAX=FETCH_VALUE(ATR,'YMAX');
   IF XMIN>=0 & XMAX<=1000 & YMIN>=250 & YMAX<=750
      THEN DO;
       XTR=0; YTR=0; DIV=1;
       END;
    ELSE DO;
       DIV=MAX((XMAX-XMIN+999)/1000,
                (YMAX-YMIN+499)/500);
       IF DIV<=0 THEN DIV=1;
       XTR=-XMIN/DIV;
       YTR=250-YMIN/DIV;
       END;
   IF DBG THEN DO;
      CALL INITIALIZE(6);
      CALL OUTPUTT('XTR='||XTR||'YTR='||YTR||
         'DIV='||DIV);
         END;
   BDF=FETCH_BDF(A);
   DO WHILE (BDF¬='');
      B=NEXT(BDF,CSEP);
      TYP=NEXT(B,' ');
      IF TYP='LINE' THEN DO;
         IF ¬ LINE THEN DO;
             CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
             I=0;  LINE='1'B;
             END;
         DO WHILE(B¬='');
            IF I<1000  THEN I=I+1;
            BL(I)=BLK*NEXT(B,' ');
            X(I)=XTR+NEXT(B,' ')/DIV;
            Y(I)=YTR+NEXT(B,' ')/DIV;
            END;
         END;
      IF TYP='POINT' THEN DO;
         IF  LINE THEN DO;
             CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
             I=0;  LINE='0'B;
             END;
         DO WHILE(B¬='');
            IF I<1000  THEN I=I+1;
            BL(I)=BLK*NEXT(B,' ');
            X(I)=XTR+NEXT(B,' ')/DIV;
            Y(I)=YTR+NEXT(B,' ')/DIV;
```

```
      END;
   END;
ELSE IF TYP='ALPHA' THEN DO;
   IF I<1000 THEN I=I+1;
   BL(I)=0;
   X(I)=XTR;
   Y(I)=YTR;
   IF LINE THEN CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
      ELSE CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
   I=0;   LINE='1'B;
   IF BLK¬=0 THEN CALL STRLAR(B,BUFADD);
   END;
ELSE IF TYP='ARC' THEN DO;
   IF ¬ LINE THEN DO;
       CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
       I=0;  LINE='1'B;
       END;
     INT= BLK*NEXT(B,' ');
     CENX=NEXT(B,' ');
     CENY=NEXT(B,' ');
     RAD=NEXT(B,' ');
     ALPHA=NEXT(B,' ');
     BETA=NEXT(B,' ');
     CALL ARC_GEN(I,INT,CENX,CENY,RAD,ALPHA,BETA);
     END;
ELSE IF TYP='TRANS' THEN DO;
     XTR=XTR+NEXT(B,' ')/DIV;
     YTR=YTR+NEXT(B,' ')/DIV;
     END;
ELSE IF TYP='UNTRAN' THEN DO;
     XTR=XTR-NEXT(B,' ')/DIV;
     YTR=YTR-NEXT(B,' ')/DIV;
     END;
ELSE IF TYP='BLANK' THEN BLK=0;
ELSE IF TYP='UNBLANK' THEN BLK=1;
END;
IF I ¬= 0 THEN DO;
  IF LINE THEN CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
    ELSE CALL PLOTP(BL,X,Y,I,1000,1,BUFADD);
  END;
I=1;
BL(I)=0;
X(I)=FETCH_VALUE(ATR,'TAILX')/DIV+XTR;
Y(I)=FETCH_VALUE(ATR,'TAILY')/DIV+YTR;
CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
CALL STRLAR('T',BUFADD);
X(I)=FETCH_VALUE(ATR,'HEADX')/DIV+XTR;
Y(I)=FETCH_VALUE(ATR,'HEADY')/DIV+YTR;
CALL PLOTL(BL,X,Y,I,1000,1,BUFADD);
CALL STRLAR('H',BUFADD);
CALL LOWER(BUFADD);
END SCAN;
```

```
SCAN1:     PROC(A,B,C,L,M);
           /* OUTPUTS A TO PRINTER WITH CHAR B AS BACKGROUND
              AND CHAR C AS DATA IN ARRAY CH(*) L X M
              A ASSUMED IN PRIM REPRESENTATION */
           DCL A CHAR(*) VAR,
               (B,C) CHAR(1),
               (L,M) FIXED BIN,
               OBG EXT BIT(1) INITIAL('0'B),
               (ALPHA,BETA,CENX,CENY,INT,RAD) FIXED BIN,
               (ATR,BDF,BB,TYP) CHAR(2000) VAR,
               CH(L,M) CHAR(1),
               (XTR,YTR,DIV,XMIN,XMAX,YMIN,YMAX,BLK) FIXED BIN,
               CSEP EXT CHAR(20) VAR,
               (I,J) FIXED BIN,
               NEXT INT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                      RETURNS(CHAR(2000) VAR),
               (FETCH_ATTR,FETCH_BDF) EXT ENTRY (CHAR(*) VAR)
                      RETURNS(CHAR(2000) VAR),
               FETCH_VALUE EXT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                      RETURNS(CHAR(2000) VAR),
               (X1,X2,Y1,Y2) FIXED BIN,
               ARC_GEN INT ENTRY(FIXED BIN,FIXED BIN,FIXED BIN,
                      FIXED BIN,FIXED BIN,FIXED BIN),
               LINE_GEN INT ENTRY(FIXED BIN,FIXED BIN,FIXED BIN,
                        FIXED BIN,FIXED BIN,CHAR(1));
               ON SUBSCRIPTRANGE BEGIN;
                   PUT LIST('SUBSCRIPT WRONG FOR CH(*)') SKIP;
                   GO TO LOOP;
                   END;
NEXT:      PROC(A,B)  RETURNS(CHAR(2000) VAR);
           /* RETURNS NEXT ELEMENT OF A DELIMINATED BY B
              DELETES FROM A */
           DCL (A,B) CHAR(*) VAR,
               TMP CHAR(2000) VAR,
               I FIXED BIN;
              IF A='' | A=' ' THEN DO;
                   A='';  RETURN('');
                   END;
              DO WHILE(SUBSTR(A,1,1)=' ');
                 A=SUBSTR(A,2);
                 END;
              I =INDEX(A,B);
              IF I=0 THEN DO;
                   TMP=A; A='';
                   END;
               ELSE DO;
                   TMP=SUBSTR(A,1,I-1);
                   A=SUBSTR(A,I+LENGTH(B));
                   END;
              RETURN(TMP);
              END NEXT;
LINE_GEN:  PROC(BLK,X1,Y1,X2,Y2,C);
           /*  GENERATES A LINE FORM (X1,Y1) TO (X2,Y2) USING CHAR
               C IN CH(L,M)--IF BLK=0 THEN NO ENTRY */
           DCL (BLK,X1,Y1,X2,Y2) FIXED BIN,
               C CHAR(1),
               (I,J) FIXED BIN;
               IF BLK=0 THEN RETURN;
               IF ABS(X2-X1)>ABS(Y2-Y1) THEN DO;
                  DO I=MIN(X1,X2) TO MAX(X1,X2);
```

```
                    J=((Y2-Y1)*I+Y1*X2-X1*Y2)/(X2-X1);
(SUBSCRIPTRANGE):   CH(I,J)=C;
                    END;
                 END;
                 ELSE IF ABS(Y2-Y1)>=ABS(X2-X1) & (Y2-Y1)¬=0 THEN DO;
                    DO J=MIN(Y1,Y2) TO MAX(Y1,Y2);
                       I=((X2-X1)*J+X1*Y2-Y1*X2)/(Y2-Y1);
(SUBSCRIPTRANGE):      CH(I,J)=C;
                       END;
                    END;
                 ELSE
(SUBSCRIPTRANGE):      CH(X1,Y1)=C;
                 END LINE_GEN;
ARC_GEN:    PROC(INT,CENX,CENY,RAD,ALPHA,BETA);
            /* GENERATES VECTORS FOR ARC */
            DCL(INT,CENX,CENY,RAD,ALPHA,BETA) FIXED BIN,
               (X1,X2,Y1,Y2,J,N) FIXED BIN,
               DELTA FLOAT BIN;
               IF INT=0 THEN RETURN;
               N=ABS(BETA-ALPHA)/5+1;
               IF N=1 THEN N=2;
               DELTA=(BETA-ALPHA)/(N-1);
               X1=(CENX+RAD*COSD(ALPHA))/DIV;     X1=X1+XTR;
               Y1=(CENY+RAD*SIND(ALPHA))/DIV;     Y1=Y1+YTR;
               DO J=2 TO N;
                  X2=(CENX+RAD*COSD(ALPHA+(J-1)*DELTA))/DIV; X2=X2+XTR;
                  Y2=(CENY+RAD*SIND(ALPHA+(J-1)*DELTA))/DIV; Y2=Y2+YTR;
                  CALL LINE_GEN(INT,X1,Y1,X2,Y2,C);
                  X1=X2;
                  Y1=Y2;
                  END;
               END ARC_GEN;
         DO I=1 TO L;
            DO J=1 TO M;
               CH(I,J)=B;
               END;
            END;
         BLK=1;
         IF M<=0 | L<=0 THEN DO;
             PUT EDIT('WRONG M AND L--M=',M,'  L=',L)
                 (SKIP,A,F(10),A,F(10));
             GO TO EXIT2;
             END;
         ATR=FETCH_ATTR(A);
         XMIN=FETCH_VALUE(ATR,'XMIN');
         XMAX=FETCH_VALUE(ATR,'XMAX');
         YMIN=FETCH_VALUE(ATR,'YMIN');
         YMAX=FETCH_VALUE(ATR,'YMAX');
         IF XMIN>=1 & XMAX<=L  & YMIN>=1 & YMAX<=M  THEN DO;
            XTR=0;
            YTR=0;
            DIV=1;
            END;
           ELSE DO;
              DIV=MAX((XMAX-XMIN+L )/L ,(YMAX-YMIN+M )/M );
              IF DIV<=0 THEN DIV=1;
              XTR=1-XMIN/DIV;
              YTR=1-YMIN/DIV;
              END;
         IF DBG THEN
```

```
                PUT EDIT('XTR=',XTR,'YTR=',YTR,'DIV=',DIV)
                        (3(SKIP,A,F(10)));
            BDF=FETCH_BDF(A);
LOOP:       DO WHILE (BDF¬='');
                BB=NEXT(BDF,CSEP);
                TYP=NEXT(BB,' ');
                IF TYP='LINE' THEN DO;
                   INT=BLK*NEXT(BB,' ');
                   X1=XTR+NEXT(BB,' ')/DIV;
                   Y1=YTR+NEXT(BB,' ')/DIV;
                   DO WHILE(BB¬='');
                      INT=BLK*NEXT(BB,' ');
                      X2=XTR+NEXT(BB,' ')/DIV;
                      Y2=YTR+NEXT(BB,' ')/DIV;
                      CALL LINE_GEN(INT,X1,Y1,X2,Y2,C);
                      X1=X2;
                      Y1=Y2;
                      END;
                   END;
                ELSE IF TYP='POINT' THEN DO;
                   DO WHILE (BB¬='');
                     INT=BLK*NEXT(BB,' ');
                     X1=XTR+NEXT(BB,' ')/DIV;
                     Y1=YTR+NEXT(BB,' ')/DIV;
                     IF INT¬=0 THEN
(SUBSCRIPTRANGE):          CH(X1,Y1)=C;
                     END;
                   END;
                ELSE IF TYP='ALPHA' THEN DO;
                  DO I=1 TO MIN(LENGTH(BB),L-XTR);
(SUBSCRIPTRANGE):    CH(XTR+I-1,YTR)=SUBSTR(BB,I,1);
                     END;
                  END;
                ELSE IF TYP='ARC' THEN DO;
                   INT=BLK*NEXT(BB,' ');
                   CENX=NEXT(BB,' ');
                   CENY=NEXT(BB,' ');
                   RAD=NEXT(BB,' ');
                   ALPHA=NEXT(BB,' ');
                   BETA=NEXT(BB,' ');
                   CALL ARC_GEN(INT,CENX,CENY,RAD,ALPHA,BETA);
                   END;
                ELSE IF TYP='TRANS' THEN DO;
                   XTR=XTR+NEXT(BB,' ')/DIV;
                   YTR=YTR+NEXT(BB,' ')/DIV;
                   END;
                ELSE IF TYP='UNTRAN' THEN DO;
                   XTR=XTR-NEXT(BB,' ')/DIV;
                   YTR=YTR-NEXT(BB,' ')/DIV;
                   END;
                ELSE IF TYP='BLANK' THEN BLK=0;
                ELSE IF TYP='UNBLANK' THEN BLK=1;
                END;
              X1=FETCH_VALUE(ATR,'TAILX')/DIV+XTR;
              Y1=FETCH_VALUE(ATR,'TAILY')/DIV+YTR;
              X2=FETCH_VALUE(ATR,'HEADX')/DIV+XTR;
              Y2=FETCH_VALUE(ATR,'HEADY')/DIV+YTR;
(SUBSCRIPTRANGE):  CH(X1,Y1)='T';
(SUBSCRIPTRANGE):  CH(X2,Y2)='H';
              DO X1=1 TO (L-1)/100+1;
                  DO I=M TO 1 BY -1;
                      PUT EDIT(' ')(SKIP,COL(20),A);
                      DO J=1+(X1-1)*100 TO MIN(L,X1*100);
                         PUT EDIT(CH(J,I))(A);
                         END;
                      END;
                  PUT EDIT(' ')(PAGE,A);
                  END;
EXIT2:       END SCAN1;
```

```
SCAN1:     PROC(A,B,C,L,M);
           /* OUTPUTS A TO TYPEWRITER WITH CHAR B AS BACKGROUND
              AND CHAR C AS DATA IN ARRAY CH(*) L X M
              A ASSUMED IN PRIM REPRESENTATION */
           DCL A CHAR(*) VAR,BUF CHAR(2000) VAR,
               (B,C) CHAR(1),
               (L,M) FIXED BIN,
               DBG EXT BIT(1) INITIAL('0'B),
               (ALPHA,BETA,CENX,CENY,INT,RAD) FIXED BIN,
               (ATR,BDF,BB,TYP) CHAR(2000) VAR,
               CH(L,M) CHAR(1),
               (XTR,YTR,DIV,XMIN,XMAX,YMIN,YMAX,BLK) FIXED BIN,
               CSEP EXT CHAR(20) VAR,
               (I,J) FIXED BIN,
               NEXT INT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                     RETURNS(CHAR(2000) VAR),
               (FETCH_ATTR,FETCH_BDF) EXT ENTRY (CHAR(*) VAR)
                     RETURNS(CHAR(2000) VAR),
               FETCH_VALUE EXT ENTRY(CHAR(*) VAR,CHAR(*) VAR)
                     RETURNS(CHAR(2000) VAR),
               (X1,X2,Y1,Y2) FIXED BIN,
               ARC_GEN INT ENTRY(FIXED BIN,FIXED BIN,FIXED BIN,
                     FIXED BIN,FIXED BIN,FIXED BIN),
               LINE_GEN INT ENTRY(FIXED BIN,FIXED BIN,FIXED BIN,
                       FIXED BIN,FIXED BIN,CHAR(1));
               ON SUBSCRIPTRANGE BEGIN;
                   CALL OUTPUT('SUBSCRIPT WRONG FOR CH(*)');
                   GO TO LOOP;
                   END;
NEXT:      PROC(A,B)  RETURNS(CHAR(2000) VAR);
           /* RETURNS NEXT ELEMENT OF A DELIMINATED BY B
              DELETES FROM A */
           DCL (A,B) CHAR(*) VAR,
               TMP CHAR(2000) VAR,
               I FIXED BIN;
              IF A='' | A=' ' THEN DO;
                   A='';  RETURN('');
                   END;
              DO WHILE(SUBSTR(A,1,1)=' ');
                 A=SUBSTR(A,2);
                 END;
              I =INDEX(A,B);
              IF I=0 THEN DO;
                   TMP=A; A='';
                   END;
               ELSE DO;
                   TMP=SUBSTR(A,1,I-1);
                   A=SUBSTR(A,I+LENGTH(B));
                   END;
              RETURN(TMP);
              END NEXT;
LINE_GEN:  PROC(BLK,X1,Y1,X2,Y2,C);
           /*  GENERATES A LINE FORM (X1,Y1) TO (X2,Y2) USING CHAR
               C IN CH(L,M)--IF BLK=0 THEN NO ENTRY */
           DCL (BLK,X1,Y1,X2,Y2) FIXED BIN,
               C CHAR(1),
               (I,J) FIXED BIN;
               IF BLK=0 THEN RETURN;
               IF ABS(X2-X1)>ABS(Y2-Y1) THEN DO;
                  DO I=MIN(X1,X2) TO MAX(X1,X2);
```

```
                        J=((Y2-Y1)*I+Y1*X2-X1*Y2)/(X2-X1);
(SUBSCRIPTRANGE):       CH(I,J)=C;
                        END;
                    END;
                    ELSE IF ABS(Y2-Y1)>=ABS(X2-X1) & (Y2-Y1)¬=0 THEN DO;
                        DO J=MIN(Y1,Y2) TO MAX(Y1,Y2);
                           I=((X2-X1)*J+X1*Y2-Y1*X2)/(Y2-Y1);
(SUBSCRIPTRANGE):          CH(I,J)=C;
                           END;
                        END;
                    ELSE
(SUBSCRIPTRANGE):          CH(X1,Y1)=C;
                    END LINE_GEN;
ARC_GEN:    PROC(INT,CENX,CENY,RAD,ALPHA,BETA);
            /* GENERATES VECTORS FOR ARC */
            DCL(INT,CENX,CENY,RAD,ALPHA,BETA) FIXED BIN,
               (X1,X2,Y1,Y2,J,N) FIXED BIN,
               DELTA FLOAT BIN;
               IF INT=0 THEN RETURN;
               N=ABS(BETA-ALPHA)/5+1;
               IF N=1 THEN N=2;
               DELTA=(BETA-ALPHA)/(N-1);
               X1=(CENX+RAD*COSD(ALPHA))/DIV;     X1=X1+XTR;
               Y1=(CENY+RAD*SIND(ALPHA))/DIV;     Y1=Y1+YTR;
               DO J=2 TO N;
                  X2=(CENX+RAD*COSD(ALPHA+(J-1)*DELTA))/DIV;  X2=X2+XTR;
                  Y2=(CENY+RAD*SIND(ALPHA+(J-1)*DELTA))/DIV;  Y2=Y2+YTR;
                  CALL LINE_GEN(INT,X1,Y1,X2,Y2,C);
                  X1=X2;
                  Y1=Y2;
                  END;
               END ARC_GEN;
         DO I=1 TO L;
            DO J=1 TO M;
               CH(I,J)=B;
               END;
            END;
         BLK=1;
         IF M<=0 | L<=0 THEN DO;
             CALL OUTPUT('WRONG M AND L--M='||M||'  L='||L);
             GO TO EXIT2;
             END;
         ATR=FETCH_ATTR(A);
         XMIN=FETCH_VALUE(ATR,'XMIN');
         XMAX=FETCH_VALUE(ATR,'XMAX');
         YMIN=FETCH_VALUE(ATR,'YMIN');
         YMAX=FETCH_VALUE(ATR,'YMAX');
         IF XMIN>=1 & XMAX<=L  & YMIN>=1 & YMAX<=M  THEN DO;
            XTR=0;
            YTR=0;
            DIV=1;
            END;
           ELSE DO;
              DIV=MAX((XMAX-XMIN+L )/L ,(YMAX-YMIN+M )/M );
              IF DIV<=0 THEN DIV=1;
              XTR=1-XMIN/DIV;
              YTR=1-YMIN/DIV;
              END;
         IF DBG THEN
            CALL OUTPUT('XTR='||XTR||'YTR='||YTR||'DIV='||DIV);
```

```
            BDF=FETCH_BDF(A);
LOOP:       DO WHILE (BDF¬='');
               BB=NEXT(BDF,CSEP);
               TYP=NEXT(BB,' ');
               IF TYP='LINE' THEN DO;
                  INT=BLK*NEXT(BB,' ');
                  X1=XTR+NEXT(BB,' ')/DIV;
                  Y1=YTR+NEXT(BB,' ')/DIV;
                  DO WHILE(BB¬='');
                     INT=BLK*NEXT(BB,' ');
                     X2=XTR+NEXT(BB,' ')/DIV;
                     Y2=YTR+NEXT(BB,' ')/DIV;
                     CALL LINE_GEN(INT,X1,Y1,X2,Y2,C);
                     X1=X2;
                     Y1=Y2;
                     END;
                  END;
               ELSE IF TYP='POINT' THEN DO;
                  DO WHILE (BB¬='');
                    INT=BLK*NEXT(BB,' ');
                    X1=XTR+NEXT(BB,' ')/DIV;
                    Y1=YTR+NEXT(BB,' ')/DIV;
                    IF INT¬=0 THEN
(SUBSCRIPTRANGE):           CH(X1,Y1)=C;
                    END;
                  END;
               ELSE IF TYP='ALPHA' THEN DO;
                 DO I=1 TO MIN(LENGTH(BB),L-XTR);
(SUBSCRIPTRANGE):   CH(XTR+I-1,YTR)=SUBSTR(BB,I,1);
                     END;
                 END;
               ELSE IF TYP='ARC' THEN DO;
                  INT=BLK*NEXT(BB,' ');
                  CENX=NEXT(BB,' ');
                  CENY=NEXT(BB,' ');
                  RAD=NEXT(BB,' ');
                  ALPHA=NEXT(BB,' ');
                  BETA=NEXT(BB,' ');
                  CALL ARC_GEN(INT,CENX,CENY,RAD,ALPHA,BETA);
                  END;
               ELSE IF TYP='TRANS' THEN DO;
                  XTR=XTR+NEXT(BB,' ')/DIV;
                  YTR=YTR+NEXT(BB,' ')/DIV;
                  END;
               ELSE IF TYP='UNTRAN' THEN DO;
                  XTR=XTR-NEXT(BB,' ')/DIV;
                  YTR=YTR-NEXT(BB,' ')/DIV;
                  END;
               ELSE IF TYP='BLANK' THEN BLK=0;
               ELSE IF TYP='UNBLANK' THEN BLK=1;
               END;
            X1=FETCH_VALUE(ATR,'TAILX')/DIV+XTR;
            Y1=FETCH_VALUE(ATR,'TAILY')/DIV+YTR;
            X2=FETCH_VALUE(ATR,'HEADX')/DIV+XTR;
            Y2=FETCH_VALUE(ATR,'HEADY')/DIV+YTR;
(SUBSCRIPTRANGE):  CH(X1,Y1)='T';
(SUBSCRIPTRANGE):  CH(X2,Y2)='H';
            DO I=M TO 1 BY -1;
               BUF='';
               DO J=1 TO L;
                 BUF=BUF||CH(J,I);
                 END;
               CALL OUTPUT(BUF);
               END;
EXIT2:      END SCAN1;
```

APPENDIX H

An Explanation of SIMPLE

The following information is taken from SIMPLE (George 1971b).

H.1 INTRODUCTION

SIMPLE is a specialized translator writing system designed to aid the implementation of an experimental graphic meta system in PL/I (George 1969a). Although intended for writing preprocessors for PL/I, experience has demonstrated that these techniques can be used to implement various specialized languages (George 1967 a,b).

SIMPLE is composed of three components: an executive, a syntax analyzer, and a semantic constructor as illustrated in Fig. H.1.

The executive reads a block of data (i.e., variable initialization) and then passes control to the syntax analyzer and then to the semantic constructor.

The syntax analyzer reads the input syntax and constructs parsing tables which are then merged as data in a general skeleton parser, in source form (PL/I); this merged program is a specific parser for the language defined by the syntax and includes a parser, automatic error recovery and error diagnostics. The syntax analyzer has two output files: the specific parser, in source form (PL/I), and diagnostics related to the syntax.

The semantic constructor reads the semantics to be associated with the previous syntax and constructs a semantic procedure compatible with the specific parser; it also has diagnostic output for errors.

The semantic constructor is defined using the syntax analyzer and a skeleton parser containing a short, hand-coded semantic procedure.

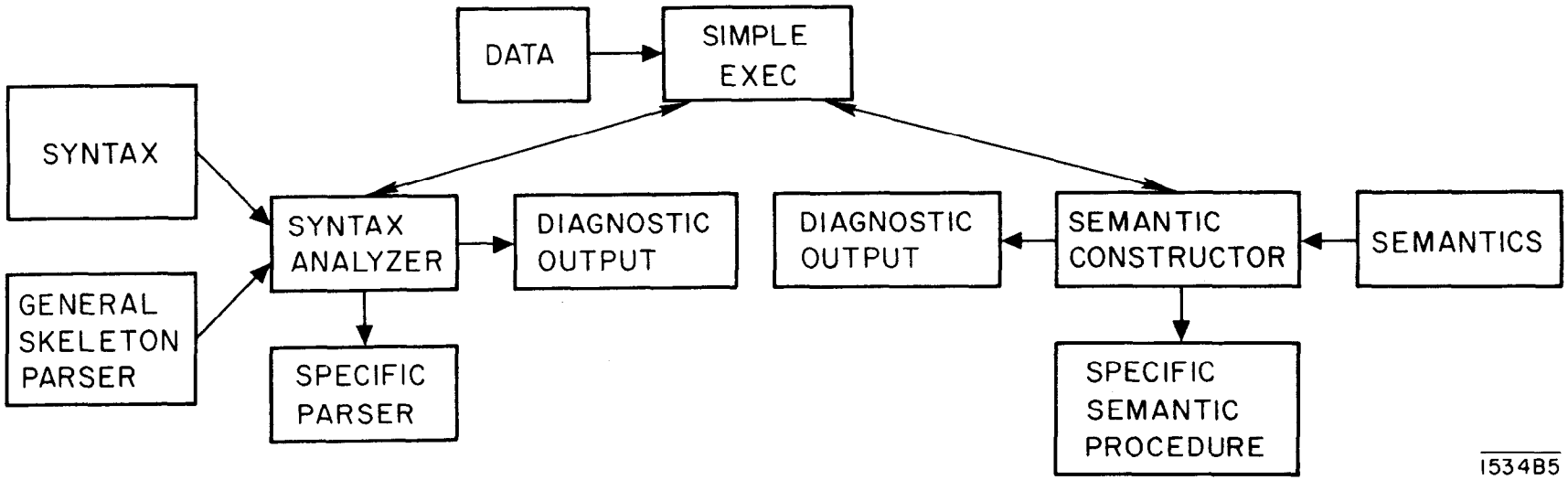


FIG. H. 1--Simple block diagram.

A language defined using SIMPLE functions as illustrated in Fig. H.2. The input text is processed by the parser which calls the semantic procedure at appropriate times. The language processor has access to two output files: a source output and a diagnostic output. Both of these files are available to the parser and the semantic procedure. A typical application would be to process input text and generate an equivalent source text (say PL/I) and error diagnostics, if any. The source output can then be compiled using a standard language processor.

H.2 Data Input to SIMPLE's Executive

The executive program initializes variables to be used by both the syntax analyzer and the semantic constructor. Any of these values may be changed by name value pairs appearing in the data file, SYNDATA (the data is read using the data directed input option in PL/I and, hence, consists of the variable name, an "=" and the value as a legal constant in PL/I). The variables are:

NAME	TYPE	DEFAULT	EXPLANATION
ERRORSCAN	CHAR(20)VAR	*END*	That symbol in the syntax which is used in error recovery. When an error is detected when parsing, all current and future text until the first occurrence of this symbol is erased.
FILE1	CHAR(8)VAR	SYNTAX	Syntax equations input file.
FILE2	CHAR(8)VAR	SPARSER	Skeleton parser input file.
FILE3	CHAR(8)VAR	PARSER	Parsing program output file.

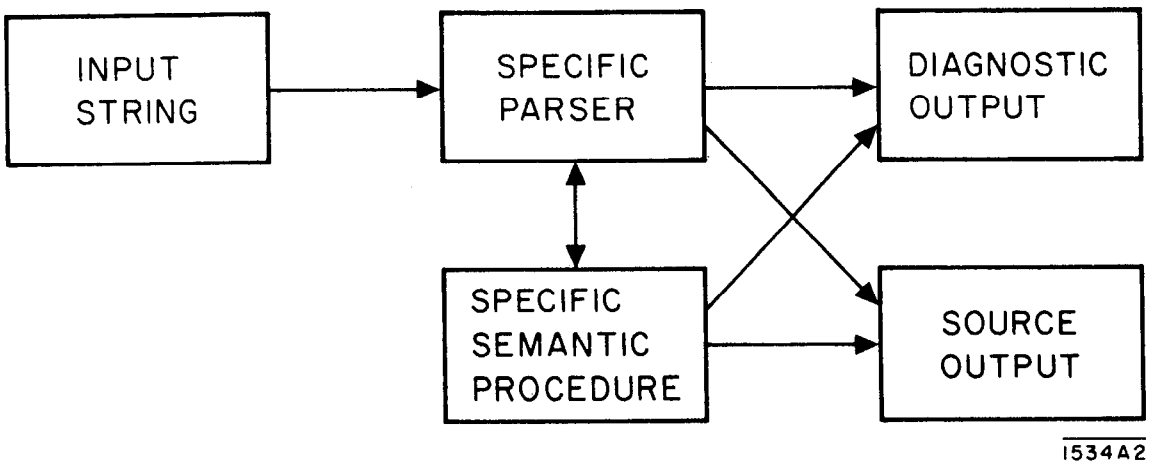


FIG. H. 2--Example simple application.

NAME	TYPE	DEFAULT	EXPLANATION
FILE4	CHAR(8)VAR	PSYNTAX	Syntax diagnostic output file.
FILE5	CHAR(8)VAR	SYNDATA	Input file for SIMPLE executive.
FILE6	CHAR(8)VAR	SEMANTICS	Semantic input file.
FILE7	CHAR(8)VAR	PSEMANT	Semantic diagnostic output file.
FILE8	CHAR(8)VAR	SEMANT	Semantic program output file.
INTEGER	CHAR(20)VAR	INTEGER	That symbol used in the syntax for an integer.
MLIM	FIXED BIN	20	Maximum number of symbols in the syntax.
MMLIM	FIXED BIN	20	Maximum number of non-basic symbols in the syntax.
NLIM	FIXED BIN	20	Maximum number of productions in the syntax.
PARSER_NAME	CHAR(8)	SEMANT	Name to be substituted for *PARSER* in FILE2; the procedure name for the parser procedure.
QUOTES	CHAR(20)VAR	"	That symbol used for quotes to force the STRING class.
RLIM	FIXED BIN	8	Maximum number of symbols on the right side in any production in the syntax.

NAME	TYPE	DEFAULT	EXPLANATION
SCAN_START	CHAR(20)VAR	*END*	That symbol not in the syntax which will restart the parsing.
SCAN_STOP	CHAR(20)VAR	*CODE*	That symbol in the syntax which, upon entry into the parsing stack, causes all input to be ignored by the parser until the symbol after SCAN_START.
SEMANT_NAME	CHAR(8)	CODE_OUT	Name to be substituted for *SEMANT* in FILE2; the name of the semantic procedure to be called by this parser.
SEND	CHAR(20)VAR	*END-SYNTAX*	Terminator for syntax.
SEQUENCE	CHAR(20)VAR	SEMANTICS	The initial symbol of the syntax; when it occurs in the stack, the parsing is terminated.
SINIT	CHAR(20)VAR	*SYNTAX*	Initiator for syntax analyzer.
SSEMANT	CHAR(20)VAR	*NO-SEMANT*	Indicates no semantics for this production.
SSEP	CHAR(20)VAR	*::=*	Separator for left-right sides.
STERM	CHAR(20)VAR	*;*	Terminator for syntax equations.

NAME	TYPE	DEFAULT	EXPLANATION
STRING	CHAR(20)VAR	STRING	That symbol in the syntax used for the string class.
SYM(*)	CHAR(20)VAR	SYM(1)='SEMANT' SYM(2)='CODA' SYM(3)='INTER-PRETATIONS' SYM(4...20)=''	Used for error recovery; those symbols which are expected to reside in the ith position of the parsing stack.
TERMINAL	CHAR(20)VAR	*END-SEMANTICS*	That symbol used to force the parsing to be completed.
WORD	CHAR(20)VAR	WORD	That symbol used in the syntax for the WORD class.

H.3 Syntax Analyzer and Parser

A simple precedence syntax analyzer was chosen for its simplicity, power and availability in a form suitable for modification. The basic analyzer was translated to PL/I from an ALGOL listing obtained from N. Wirth (Wirth and Weber, 1966 a, b). Many sections were modified to take advantage of features of PL/I. The changes to the analyzer are:

1. The input section was modified to be free field and to mark productions with no semantics;
2. Maximum number of right part elements is variable;
3. Three terminal classes are recognized rather than two (this holds in the parser also);
4. The output section inserts PL/I declarations into a skeleton parser rather than punching tables.

H.3.1 Input Conventions for the Syntax Analyzer

The input for the syntax analyzer (i.e., the productions) is contained in a file whose default name is SYNTAX (setting this name is explained in Section H.2). The formal definition of the syntax is:

⟨SYNTAX⟩ ::= ⟨SINIT⟩ ⟨PRODUCTIONS⟩ ⟨SEND⟩

⟨PRODUCTIONS⟩ ::= ⟨PRODUCTION⟩

::= ⟨PRODUCTIONS⟩ ⟨SYSTEM⟩ ⟨PRODUCTION⟩

⟨PRODUCTION⟩ ::= ⟨LEFT-PART⟩ ⟨SSEP⟩ ⟨RIGHT-PART⟩

::= ⟨LEFT-PART⟩ ⟨SSEP⟩ ⟨RIGHT-PART⟩ ⟨SSEMANT⟩

⟨LEFT-PART⟩ ::= ⟨SYMBOL⟩

⟨RIGHT-PART⟩ ::= ⟨SYMBOL⟩

::= ⟨RIGHT-PART⟩ ⟨SYMBOL⟩

⟨SYMBOL⟩ ::= any string excluding blanks

The default values are:

⟨SINIT⟩ = *SYNTAX*

⟨SEND⟩ = *END-SYNTAX*

⟨STERM⟩ = *;*

⟨SSEP⟩ = *::=*

⟨SSEMANT⟩ = *NO-SEMANT*

The input is free field card images using blanks or a new card to separate symbols; only the first 20 characters of a symbol are used.

In actual use there are two additional limits:

1. Upper limit on number of productions;
2. Upper limit on number of symbols in any right part.

If more productions than the limit of productions are used, then those productions between the limit less one and the last productions are lost; similarly,

for more symbols in the right part than the limit. Note that both of these are input parameters to SIMPLE (Section H.2).

If the left part has more than one symbol then the last symbol in the left part is used.

H.3.2 Syntax Analyzer Output

In addition to inserting the necessary declarations and initialization into the skeleton parser, the syntax analyzer generates a file (FILE4 whose default name is PSYNTAX) which contains information about the syntax and any errors. This output consists of:

1. Productions - The productions are numbered in the order that they are read in and this number is used to select the applicable portion of the semantic procedure.
2. Basic and nonbasic symbols - The basic and nonbasic symbols are assigned a unique number.
3. KEY and PRTB tables (Shaw 1966a p. 194) - These are used by the parser in determining the production number and the left part of the production of a reducible substring. "KEY(i) represents for the ith symbol (i corresponds to the number assigned in 2) the index in the production table PRTB, where those productions are listed whose right part string begins with the ith symbol. For each production, the right part is listed without its leftmost symbol, followed by the production number (negative) and the left part symbol of the production. The end of the list of productions referenced via KEY(i) is marked with a 0 entry in PRTB." If a production has no semantics then the production number in PRTB is adjusted to be out of range (by the number of productions).

4. Right and left symbol sets - These are sometimes useful in removing conflicts.

5. PRECEDENCE Matrix - Two symbols x and y are related (either $x=y$, $x < y$, $x > y$ or no relation) by the entry in the ith row (where i is the number corresponding to x) and j^{th} column (j corresponding to y) of the matrix.

6. DIAGNOSTICS

   a. For a correct syntax

      NO PRECEDENCE VIOLATIONS OCCURRED

   b. For an incorrect syntax

      1. PRECEDENCE VIOLATIONS OCCURRED

         HINTS REGARDING PRECEDENCE VIOLATION

         The most recent production number which causes a violation followed by the two symbols separated by the two relations.

   c. Incorrect input file

      ***** ENDFILE SYNTAX INPUT - NO

      followed by the value of SEND.

      SEND missing generally causes no problems. If there is no additional syntax output, then the symbol SINIT was never encountered.