SLAC-PUB-664
September 1969
(MISC)

# ON THE INTERACTIVE GENERATION AND INTERPRETATION OF ARTIFICIAL PICTURES*

Alan C. Shaw
Cornell University**
Ithaca, New York 14850

(For Presentation at the 1969 ACM/SIAM/IEEE Conference on Mathematical and Computer Aids to Design, Anaheim, California, October 26-30, 1969.)

*This work was supported, in part, by the U.S. Atomic Energy Commission while the author was employed at the Stanford Linear Accelerator Center, Stanford University, Stanford, California during the summer of 1969.

**Department of Computer Science.

## ABSTRACT

This paper presents the design of a general set of user facilities for the on-line generation and interpretation of pictures. We are concerned with the interactive graphics language, the command and control language, as opposed to the language used to implement it. The intent is to provide a superior alternative to the present mode of operation which usually involves the definition of a new command and control language and associated translator for each new application (and machine).

Pictures are not drawn in the conventional manner using a light pen or stylus and tablet. Instead, they are generated by executing or evaluating an algebraic-like string description in a picture description language. Graphics entities are described in the space that is most natural to the user — a virtual picture space — and then mapped into the real picture space of the display device. Attributes may be specified and assigned to picture parts in an arbitrary manner. Pictures are interpreted by parsing them according to given grammars and by performing computations with pictures as arguments. An on-line computational facility is also used for computing complex attributes and constraints. The system is illustrated by examples in the drawing and execution of flow charts, and the generation and analysis of series/parallel resistance networks.

## TABLE OF CONTENTS

## I. INTRODUCTION

The potential benefits of interactive graphical communication between humans and computers were first dramatically brought to the attention of the general technical community in the Sketchpad system of I. Sutherland (1963). In a typical design situation, a user sits in front of a console equipped with both graphical and textual input/output devices as well as a variety of buttons and switches. A model of the entity being designed is incrementally specified and communicated to the machine through the input devices; the computer, in turn, is programmed to interpret the user input by analyzing his specifications and displaying some abstraction of the design. This is a feedback process with the user continually modifying his design until the results are satisfactory. For our purposes, the key feature of this type of interaction is the use of pictures as the basic medium of communication. We call these _artificial_ pictures since they do not appear in nature but are constructed by men and machines. Simply stated, our goal is to develop more convenient ways to specify and interpret artificial pictures in the above type of environment. This paper is intended as a contribution towards that goal.

It has been argued by us and others that most graphics applications involve _both_ picture generation and analysis (or interpretation) — what is generated must be analyzed and vice versa (Miller and Shaw, 1968b; Kulsrud, 1968). Our approach is to use a common picture description scheme for both. The purpose of picture analysis is then to _derive_ a description while in picture generation, descriptions are _executed_. This idea forms the basis for our thoughts on an interactive graphics system and will be further developed.

In the next section, we discuss a number of desirable user facilities for interactive picture generation and interpretation. Present graphics systems are then briefly and critically examined. A new and somewhat novel graphics system design is presented in the remainder of the paper. Drawing is accomplished, not by moving a lightpen or stylus, but by expressing in an algebraic-like manner a precise description of the picture. The object is to provide a drawing language which is unambiguous, device independent, and avoids the awkwardness of a multitude of switches and buttons. Pictures are tested for well-formedness and syntactically interpreted by parsing them according to given picture grammars. Computations over pictures may be interactively specified and performed during drawing and analysis. We illustrate the system by examples in flow chart design and execution, and simple electric circuit design. While the system has not yet been implemented, the feasibility of its major components has been confirmed by several related research efforts over the last few years undertaken by W. F. Miller, J. E. George, and the author (Miller and Shaw, 1968a; Shaw, 1968, 1969a, b; George and Miller, 1968; George, 1968).

## II. PICTURE PROCESSING FACILITIES FOR INTERACTIVE DESIGN

We take the point of view of a user and briefly discuss some of the drawing and interpretation facilities that an interactive computer system should provide. A great deal of the following also applies to a non-interactive situation.

What is required in order to usefully generate pictures? First, and most obvious, the picture must be described to a computer. Generally, a picture can be defined as a set of primitive components and the relations that they satisfy. If the primitives are completely defined geometrically then these relations are implicit; conversely, describing the relationships among the primitives serves to more completely define each individual primitive. Both methods of definition are useful. Given a set of basic primitives, such as points and curve segments, more complex entities should be definable in terms of these. One wishes to identify higher level structures by assigning names to them. (The distinction between primitives and higher level structures is that the user is interested in manipulating and analyzing the components of the latter but not the former.) Assigning arbitrary names to pictures should also be part of a general attribute assignment facility; for example, the attributes of a resistor might include its resistance, tolerance, and manufacturer. The above facilities suffice for statically describing a drawing. In an interactive mode, we also need to dynamically modify pictures. Thus, we include a capability for deletion, insertion, and merging, as well as for common mathematical transformations of translation, rotation, and scale change. A user finds it most convenient to define and manipulate pictures in terms of operations in a virtual picture space — the space that is most natural for the problem. Display of a picture then requires a mapping of the virtual space to the real space of the display device. The ability to dynamically specify this mapping is useful for "zooming" and moving through large pictures. Finally, unless the system is to be a toy, commands for picture storage and retrieval must be available.

We distinguish between two types of picture interpretation. The first can be viewed as classification and description. In any particular application, there is a set (possibly infinite) of "well-formed" pictures to which the generated pictures are restricted. At the most basic level, any picture specified by a user should be analyzed to determine whether that picture is a member of the set; if not, presumably the user has made an error. Additionally, it is often desirable to classify the picture and describe it in terms, other than the description given by the user when generating it. For example, in an electric circuit design application, it might be stipulated that all well-formed circuits contain certain components, such as a "ground" or a "voltage source." The designer might describe a circuit in terms of its components and connections but would want the system to impose a structure on it and produce a description of the form "This is a circuit of type $X_0$ composed of subcircuits of types $X_1$, $X_2$, ... which in turn consist of ... " This kind of picture interpretation is really the results of a syntactic analysis.

The second type of interpretation is a more conventional one and implies a facility to perform arbitrary computations over pictures and their attributes. We do not design something just to look at a picture of it but want to meet a given set of specifications in some optimal manner. We will call this semantic analysis. We might compute and display, for example, the response of an electrical network or the results of executing a flow chart. In addition, it is useful to have an on-line computational ability while generating pictures in order to determine complex drawing attributes and constraints.

## III. TECHNIQUES OF USER INTERACTION AND APPLICATIONS IMPLEMENTATION

The most common hardware interface consists of a cathode-ray tube (CRT) display for graphics output and a lightpen, alphanumeric keyboard, and function keyboard for input (e.g., Appel *et al.*, 1968). User communication with the computer occurs through interrupts generated by the input devices; these are interrogated by systems programs which, in turn, usually invoke applications programs. The latter are primarily translators for the user input language — commonly called a command and control language. "Sentences" or programs in this language are not strings over alphanumeric characters but are comprised of sequences of lightpen "points," keys on the function keyboard, and textual information.

We illustrate a graphics part of a typical command and control language by a simple example for drawing line segments on a CRT using a lightpen and one button on a function keyboard:

"1. press button to start pen tracking;
2. track pen to starting point of line;
3. press button to fix starting point;
4. track pen to end point;
5. press button to fix end point and stop tracking."

(Newman, 1968, p. 47)

A "rubber band" series of line segments is displayed as the user draws during step 4; at step 5, the rubber band is eliminated and a straight line segment is displayed between the starting and end points.

In general, picture descriptions for generation are specified to the computer using all the input devices in various combinations. This same mechanism is used for picture interpretation; the input device interrupts lead to applications programs which perform the desired analysis. Each application usually involves the definition of a new command and control language and associated translator.

The lightpen is currently being replaced in many installations by a tablet and stylus to allow a more natural way of drawing; in these systems, it is convenient to permit direct hand-drawn input since pattern recognition results may immediately be displayed on the CRT for possible correction (e.g., Sutherland and Forgie, 1969). Various other input devices, such as toggle switches and shaft encoders, have also been employed in addition to the above.

Since the philosophy has been that each application requires its own language, much effort has been devoted to the design of general purpose data structures and programming systems within which command and control languages may be defined and implemented. The simplest approach has been to add graphics subroutines to some general purpose language, for example FORTRAN (Rully, 1968). For more sophisticated applications, many data structure oriented languages have been developed; these range from linked-list macro languages such as CORAL (Roberts, 1964) to associative extensions of higher level languages, such as LEAP (Feldman and Rovner, 1969). More recently, translator writing systems for graphics languages have been devised (George, 1969; Kulsrud, 1968; Pankhurst, 1968).

We are critical of several aspects of the present mode of operation. Consider the manner in which pictures are interactively specified for generation. The drawing language is defined anew for each application in some ad hoc manner and, with exceptions, involves the awkward use of an often bewildering array of input devices. General purpose programming languages are used to define computations interactively; it seems that general purpose picture languages that allow the interactive drawing of pictures should similarly be available. A similar argument follows for picture interpretation. We need general purpose on-line languages for describing well-formedness of pictures and computations over pictures. The remainder of this paper offers a proposal in this direction.

## IV. A NEW APPROACH

Our approach to interactive graphical communication is based on the use of a formal picture description scheme. A user describes drawings and their intended interpretation in terms of a general picture description language, picture grammars, and functions with descriptions as arguments; the computer displays graphical entities by executing their descriptions, and interprets them by both parsing pictures according to given grammars to yield a description and performing computations over them. Our picture description scheme is a string language and, thus, we do not see any need for a variety of input devices; either an alphanumeric keyboard or a tablet and stylus is the only input device required.

While we are dubious about the practical value of on-line flow charting systems, the subject matter is familiar to a large group and thus serves as a good example for expository purposes. Flow charts of the type illustrated in Fig. 1 will be used as one example in the following sections; we will describe how they might be drawn and executed. The second example will be the design of simple series/parallel resistance networks, using a schematic representation as in Fig. 2. These networks will be interpreted by computing their equivalent resistance.

## V. PICTURE GENERATION

A picture will be defined as a real valued function $f(\underline{x})$ of two real variables, $\underline{x} = (x_1, x_2)$ (Rosenfeld, 1969); $\underline{x}$ is interpreted as a point in a two-dimensional Euclidean space. We will consider only pictures in the plane but most of the scheme carries over directly into higher dimensions. The description language presented below is a variation and extension of the "PDL" notation given in Shaw (1968, 1969a) and Miller and Shaw (1968a).

### A. Picture Primitives

Basic primitives are divided into disjoint sets of patterns; each set will be represented by a quadruple:

$$<n, f(\underline{x}), \underline{t}, \underline{h}> ,$$

where n is a name given to the set, the picture $f(\underline{x})$ is a distinguished primitive in the set called its normal form, and $\underline{t}$ and $\underline{h}$ are two distinguished points in $f(\underline{x})$, called the tail and head respectively. We alternately denote the set of pictures in the primitive class named n as $\mathscr{P}(n)$. $\mathscr{P}(n)$ is defined as all <u>translations</u> over the plane of $f(\underline{x})$; i.e., $\mathscr{P}(n) = \{f(\underline{x} - \underline{c}) \mid \underline{c} \text{ in } R^2\}$. R is the set of real numbers. $f(\underline{x} - \underline{c})$ has tail at $\underline{t} + \underline{c}$ and head at $\underline{h} + \underline{c}$.

<u>Examples</u>

The value of $f(\underline{x})$ at each $\underline{x}$ (the "grey level") will be restricted to the set $\{0, 1\}$ but could easily have other values.

1.(a) The primitive class "horizontal line segments of unit length" can be defined as:

$$< \ell,\ f(\underline{x}),\ (0,0),\ (1,0) > ,$$

where $f(\underline{x}) = \begin{cases} 1 \text{ for } 0 \le x_1 \le 1,\ x_2 = 0 \\ 0 \text{ elsewhere} . \end{cases}$

(b) "Blank" line segments can similarly be defined:

$$< b,\ f(\underline{x}),\ (0,0),\ (1,0) > ,$$

where $f(\underline{x}) = 0$ everywhere.

2.(a) The set of all "points" in the plane is defined:

$$< p,\ f(\underline{x}),\ (0,0),\ (0,0) > ,$$

where $f(\underline{x}) = \begin{cases} 1 \text{ for } \underline{x} = (0,0) \\ 0 \text{ elsewhere} . \end{cases}$

(b) Blank points are described:

$$< \lambda,\ f(\underline{x}),\ (0,0),\ (0,0) > ,$$

where $f(\underline{x}) = 0$ everywhere.

3. A circle primitive may be given as:

$$< c,\ f(\underline{x}),\ (0,-1),\ (0,1) > ,$$

where $f(\underline{x}) = \begin{cases} 1 \text{ when } x_1^2 + x_2^2 = 1 \\ 0 \text{ elsewhere} . \end{cases}$

Concatenations of primitive elements can only occur at their tail and head points. It is convenient to represent a primitive by an edge of a directed graph, labelled by its class name and "pointing" from its tail to head "node." We expand this definition of primitives in section V.C to handle an arbitrary number of concatenation or attachment points and parameters.

Scale changes are described by a magnification operator M[s], s a real number. For the primitive set n,

$$\mathscr{P}(M[s]n) = \left\{ f(\tfrac{1}{s}(\underline{x} - \underline{c})) \mid \underline{c} \in R^2 \right\}.$$

Rotations are handled in a similar manner by the operator R $[\theta]$, $\theta$ a real number:

$$\mathscr{P}(R[\theta]\ n) = \{ f((\underline{x} - \underline{c})\ A(\theta) \mid \underline{c} \in R^2 \},$$

where $A(\theta) = \begin{bmatrix} \cos\theta\pi & -\sin\theta\pi \\ \sin\theta\pi & \cos\theta\pi \end{bmatrix}$ .

The tail and head of $f(\frac{1}{s}(\underline{x} - \underline{c})\ A(\theta))$ are at $s\ (\underline{t} + \underline{c})\ \tilde{A}$ and $s\ (\underline{h} + \underline{c})\ \tilde{A}$ respectively, where $\tilde{A}(\theta) = A(-\theta)$. M and R operating on the line segment primitive $\ell$ change its length and orientation respectively. M[s] applied to the circle c results in a change of radius to s while R $[\theta]$ has the sole effect of rotating c's tail and head through an angle $\theta$.

To display anything, the computer must first be given a mapping function from the virtual space of the user's pictures to the real space of the display device. We will assume that both spaces are bounded by rectangles and specify the mapping by the command:

$$\underline{\text{Virtualspace}}\ (\underline{u}_1, \underline{u}_2)\#$$

where $\underline{u}_1$ is the lower left corner coordinates of a bounding rectangle in virtual space and $\underline{u}_2$ is the upper right corner coordinates.*

*At this point, we are not concerned with the form of the command and control language; more convenient abbreviations will undoubtedly be used.

Pictures are generated by the computer execution of the command:

Evaluate (S)#

where S is a picture description.* Assume a clear display.

If n is the name of a primitive class, Evaluate (n)# will first cause the normal form of n to be translated so that its tail is at the origin of virtual picture space; this picture is then mapped according to the most recently called Virtualspace function and the result displayed. Similarly, Evaluate (S) # where $S = M[s]n$ or $S = R[\theta]n$ or $S = M[s]R[\theta]n$ will cause the appropriately transformed and mapped primitive to be displayed. Note that the parts of a picture not contained within the virtual space rectangle will not be displayed. A picture described by S is no longer displayed when the computer executes the input command:

Delete (S) #

A generation function will be associated with each display device and primitive class. The function corresponding to a primitive class named n and a display device d will produce the display file commands for generating any member of $\mathscr{P}(M[s]\ R[\theta]\ n)$, for arbitrary s and $\theta$, on device d.

## B. Picture Description and Evaluation

A picture will consist of a connected set of primitives and can be represented as a connected graph with labelled edges, where each edge denotes a primitive and the graph connectivity is defined by the tail/head connectivity of the primitives. A picture will be described relative to the origin of the picture coordinate system; we assume that the tail or head of at least one of its primitives is identical with the origin.

The connectivity and mathematical transformation of pictures is specified by a string in the PDL picture description language. Any "sentence" S in PDL

*Use of the term "evaluate" in the context of evaluating a picture description is due to J. E. George.

can be generated by the following syntax:

$$S \rightarrow E \mid E\ \emptyset_b\ S \mid \emptyset_u\ S \mid TS \mid S\_L$$

$$\emptyset_b \rightarrow \oplus \mid \ominus \mid \otimes \mid \circledast$$

$$\emptyset_u \rightarrow \sim \mid /$$

$$E \rightarrow N \mid N(ARG) \mid (S)$$

$$N \rightarrow \{\text{name of a primitive, subprimitive, or higher level structure}\}$$

$$ARG \rightarrow V \mid V,\ ARG$$

$$V \rightarrow \{\text{variable name}\} \mid \{\text{constant}\}$$

$$T \rightarrow M[V] \mid R[V]$$

$$L \rightarrow \{\text{label identifier}\} \mid \{\text{label identifier}\}\_L$$

<u>Example</u>

An arbitrary rectangle may be described by the following PDL expression:

$$(M[\text{vside}]\ R[0.5]\ \ell \oplus M[\text{hside}]\ \ell) \circledast (M[\text{hside}]\ \ell \oplus M[\text{vside}]\ R[0.5]\ \ell)$$

where $\ell$ is the line segment primitive defined earlier.

Let $\mathscr{P}(S)$ be the set of all pictures described by the PDL expression S; all members of $\mathscr{P}(S)$ have the same primitive set, the same connectivity, and have a tail and head defined by S. The operators generated by $\emptyset_b$ are <u>binary</u> <u>concatenation</u> <u>operators</u>. If S is of the form $S_1\ \emptyset\ S_2$ or $(S_1\ \emptyset\ S_2)$, then each picture $f \in \mathscr{P}(S)$ consists of two subpictures $f_1 \in \mathscr{P}(S_1)$ and $f_2 \in \mathscr{P}(S_2)$ and has the properties:

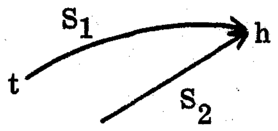
1. tail (f) = tail $(f_1)$, head (f) = head $(f_2)$.

2.(a) If $\emptyset = \oplus$ then head $(f_1)$ <u>cat</u> tail $(f_2)$, where <u>cat</u> means "is concatenated onto"; this may be represented by the graph

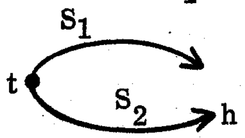
$$\begin{tikzcd} t \arrow[r, bend left, "S_1"] & {} \arrow[r, bend left, "S_2"] & h \end{tikzcd}$$

where t and h are the tail and head nodes.

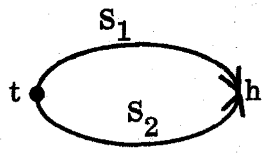
(b) If $\emptyset = \ominus$ then head $(f_1)$ cat head $(f_2)$:



(c) If $\emptyset = \otimes$ then tail $(f_1)$ cat tail $(f_2)$:



(d) If $\emptyset = \circledast$ then both tail $(f_1)$ cat tail $(f_2)$ and head $(f_1)$ cat head $(f_2)$:

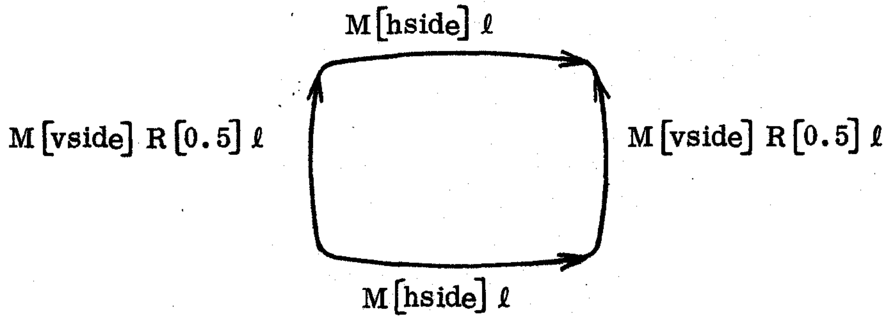


The meaning of these operators may be alternately given by defining $\mathscr{P}(S)$; for example:

$$\mathscr{P}(S_1 \oplus S_2) = \{ f(\underline{x} - \underline{c}) \mid \underline{c} \in R^2, f(\underline{x}) = f_1(\underline{x}) \cup f_2(\underline{x}),$$

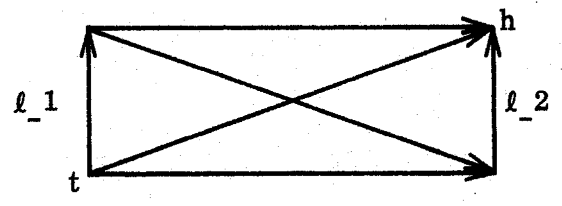
$$f_1 \in \mathscr{P}(S_1),\ f_2 \in \mathscr{P}(S_2),\ \text{head}\ (f_1)\ \underline{\text{cat}}\ \text{tail}\ (f_2) \}$$

In the absence of parentheses, association of expressions is from left to right. The rectangle in the first example has the connectivity:



We will use the notation tail (S) and head (S) to denote both the tail and head of any picture described by S, and the tail and head nodes of the connectivity graph.

The unary operator ~ is a tail/head reverser such that tail (S) = head (~ S) and head (S) = tail (~ S). Label identifiers associated with any expression allow the unique identification of the elements of the expression; in conjunction with the superposition operator /, arbitrary paths through the picture or graph may be retraced. For example we describe a rectangle with interior diagonals as follows:



$$(M[\text{vside}]\, R[0.5]\, \ell\_1 \oplus M[\text{hside}]\, \ell) \circledast (M[\text{hside}]\, \ell \oplus M[\text{vside}]\, R[0.5]\, \ell\_2) \circledast$$
$$M[s_1]\, R[\theta_1]\, \ell \circledast (/\ell\_1 \oplus M[s_2] R[\theta_2]\, \ell \oplus /\ell\_2)$$

Assuming that vside and hside are constants, $s_1$, $\theta_1$, $s_2$, and $\theta_2$ will be computed by the system to satisfy the constraints implied by the concatenation operators (Constraint satisfaction will be examined in more detail in section V.F) /ℓ_1 and /ℓ_2 indicate a retracing over the primitives M[vside] R[0.5] ℓ_1 and M[vside] R[0.5] ℓ_2; note that the labels uniquely identify the two primitives so that M and R need not appear in the retracing description.

For / to make any sense as a superposition or retracing operator in a PDL expression, it is necessary that its primitive operands also appear once and only once outside the scope of /. All expressions using / that satisfy this condition are referred to as valid PDL expressions. Labels are not mandatory for retracing purposes if the primitives are already uniquely identified by name or transformation; in the above example, if the appropriate M and R transformations were repeated in the retracing operations, the labels would not be required. However, labels are useful in other contexts as discussed in following sections.

Associating a label with an expression is equivalent to labeling each primitive within the expression; e.g., $(a\_i \oplus b)\_j$ has the same interpretation as $(a\_i\_j \oplus b\_j)$.

Any connected set of primitives may be described in the PDL notation; more abstractly, any connected graph with labelled edges can be described. The tail and head may be moved anywhere in the picture or graph for attaching new structures by suitable "path" retracing.

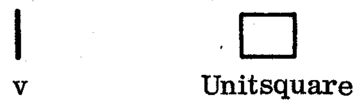
The scale change and rotation operators M and R are <u>linear</u> with respect to PDL expressions, e.g., $\mathscr{P}(M[s](S_1 \oplus S_2)) = \mathscr{P}(M[s]S_1 \oplus M[s]S_2)$. Thus, transforming an entire expression is equivalent to applying the same transformation to each of its primitives.

Sets of pictures satisfying the same PDL expression may be named with an assignment operator ":=" .

<u>Examples</u>

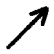
1. $v := R[0.5]\ \ell$ #

   Unit square := $(\ell \oplus v) \circledast (v \oplus \ell)$ #

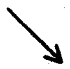


2. d1 := $R[.33]\ell$ #

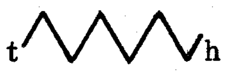
   dp := $M[2]$ d1 #



   dm := $R[-0.67]$ dp #



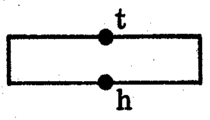
   Res := (d1 $\oplus$ dm $\oplus$ dp $\oplus$ dm $\oplus$ dp $\oplus$ dm $\oplus$ d1) #



Res will denote a resistor in our network example.

3. A function box for our flow chart example can be described:

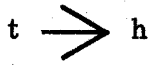
func := $M[2]((\ell \oplus \sim v \oplus \sim \ell) \circledast (\sim \ell \oplus \sim v \oplus \ell))\#$



Note the locations of the tail and head which permit a natural attachment of entering and exiting line segments.

4. The basic flow chart arrow is:

ar := $\ell \oplus (R[0.83]\, \ell \otimes R[-0.83]\, \ell \otimes \lambda)\#$



Name assignment with parameters is handled in a similar manner. The class of all rectangles may be described incrementally:

V(vside) := $M[\text{vside}]\; R[0.5]\; \ell\, \#$

H(hside) := $M[\text{hside}]\; \ell\, \#$

Rectangle (vside, hside) := $(V(\text{vside}) \oplus H(\text{hside})) \circledast (H(\text{hside}) \oplus V(\text{vside}))\#$

The command Evaluate (S) # , where S is a PDL expression, will result in (a) the formation in virtual picture space of one member f of $\mathscr{P}(S)$ such that:

(i) the normal forms of all primitives in S (possibly transformed by M and R if so described) with tail at tail (S) appear with tail at the picture origin, and

(ii) the normal forms of the remaining primitives in S are translated (and possibly transformed by M and R) to satisfy the tail/head constraints given by the concatenations described in S. (More general constraints are treated in section V.F.)

(b) f is then mapped to real picture space using the most recently invoked Virtualspace function, and displayed. ((a)(i) assumes a clear display initially.)

The command is invalid if more than one picture or no picture can be formed in (a). This is possible if S is not a valid PDL expression or if the constraints implied by the $\circledast$ operator cannot be satisfied uniquely.

## C. Complex Primitives

We generalize the definition of primitives to include complex primitives with an arbitrary number of attachment points and parameters. A complex primitive class is represented by a 5-tuple:

$$< n(\underline{a}),\ f(\underline{x};\underline{a}),\ \underline{t}(\underline{a}),\ \underline{h}(\underline{a}),\ \text{subprimitives} > ,$$

where n is the name of the set, $\underline{a}$ is a parameter list $(a_1, a_2, \ldots, a_k)$ (possibly empty) with each $a_i$ restricted to a given set of admissible values $T_i \subseteq R$, $f(\underline{x};\underline{a})$ specifies a set of normal form pictures $\{f(\underline{x};\underline{a}) \mid \underline{a} \in T_1 \times T_2 \times \ldots \times T_k\}$, $\underline{t}(\underline{a})$ and $\underline{h}(\underline{a})$ are the tail and head coordinates of $f(\underline{x};\underline{a})$, and subprimitives is a list of subprimitives of n.

Subprimitives = $((n_1, \underline{t}_1(\underline{a}), \underline{h}_1(\underline{a})), (n_2, \underline{t}_2(\underline{a}), \underline{h}_2(\underline{a})), \ldots, (n_m, \underline{t}_m(\underline{a}), \underline{h}_m(\underline{a})))$ ,

where $n_i$ is the name of the ith subprimitive and $\underline{t}_i(\underline{a})$, $\underline{h}_i(\underline{a})$ are the tail and head coordinates of $n_i$.

n and each $n_i$ is represented by a labelled directed edge of a graph; with no loss of generality, we require that this graph be connected. $\underline{t}, \underline{h}$ and all $\underline{t}_i, \underline{h}_i$ are permissible concatenation points. It is then possible to describe the primitive substructure as a PDL expression and thus "reach into" any attachment point of the primitive. In a manner analogous to that for basic primitives, $\mathscr{P}(n(\underline{a}))$ denotes all translations over the plane of elements of $f(\underline{x};\underline{a})$; i.e.,

$\mathscr{P}(n(\underline{a})) = \{f(\underline{x}-\underline{c};\underline{a}) \mid \underline{c} \in R^2, \underline{a} \in T_1 \times T_2 \times \ldots \times T_k\}$ . $\mathscr{P}(M[s]\,n(\underline{a}))$ and $\mathscr{P}(R[\theta]\,n(\underline{a}))$ are similarly defined.

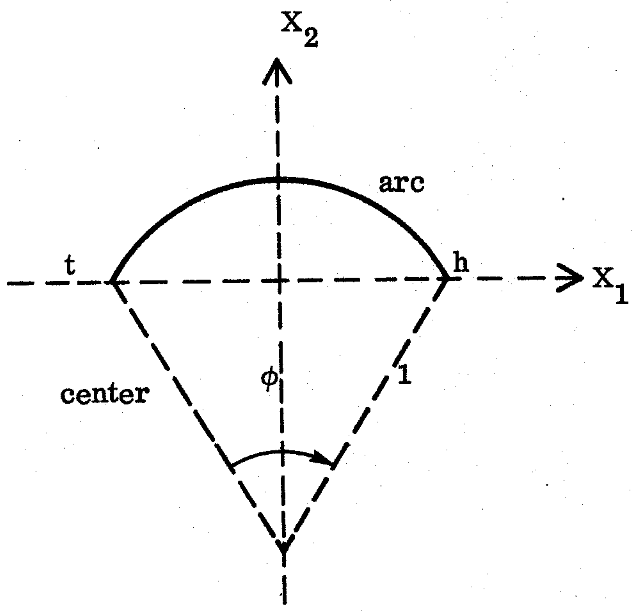
Examples

1. The primitive class "arc" subtending an angle of $\theta\pi$ radians can be defined:

$$\langle \text{arc}(\theta), f(\underline{x};\theta), (-\sin\frac{\phi}{2}, 0), (\sin\frac{\phi}{2}, 0), ((\text{center}, (-\sin\frac{\phi}{2}, 0), (0, -\cos\frac{\phi}{2}))\rangle ,$$

$$\text{where } f(\underline{x};\theta) = \begin{cases} 1 \text{ if } x_1^2 + (x_2 + \cos\frac{\phi}{2})^2 = 1 \text{ and } x_2 \geq 0 \\ 0 \text{ elsewhere} \end{cases}$$

$0 \leq \theta \leq 2$ is the region $T_1$, and $\phi = \theta\pi$ .



$$\begin{tikzcd} t \arrow[r, "arc"'] \arrow[dr, "center"'] & h \\ & \text{graph} \end{tikzcd}$$

M[s] arc($\theta$) has the effect of changing the radius from 1 to s but keeping $\theta$ constant. To concatenate the picture described by $S_1$ onto the head of arc and the picture described by $S_2$ onto the head of center, the PDL expression (arc $\oplus$ $S_1$) $\otimes$ (/center $\oplus$ $S_2$) is sufficient.

$$\begin{tikzcd} \cdot \arrow[r, "arc"] \arrow[dr, "center"'] & \textcircled{S_1} \\ & \textcircled{S_2} \end{tikzcd}$$

Use of arc automatically makes the substructure available. Evaluate (arc (0.25)) # would cause the display of an arc subtending an angle of $\pi/4$ radians.

2.(a) A point at relative location (u, v) can be described in terms of the primitive set:

$$<Q(u,v),\ f(\underline{x};(u,v)),\ (0,0),\ (u,v),\ \Lambda>$$

where $f(\underline{x};(u,v)) = \begin{cases} 1 \text{ if } x_1 = u \text{ and } x_2 = v \\ 0 \text{ elsewhere} \end{cases}$

and $\Lambda$ denotes that the field is empty.

Alternately, we could use a PDL expression to define Q(u, v) as follows:

$$Q(u,v) = (M[u]\,b \oplus M[v]\,R[0.5]\,b \oplus p) \quad .$$

Similarly, we define a blank point $\alpha(u,v)$ at relative location (u, v).

(b) A line segment with one endpoint (the head) at relative location (u, v) to the other (the tail) is a member of the set:

$$<L(u,v),\ f(\underline{x};(u,v)),\ (0,0),\ (u,v),\ \Lambda> \quad ,$$

where $f(\underline{x};(u,v)) = \begin{cases} 1 \text{ if } vx_1 - ux_2 = 0 \\ \quad \text{and } x_1 \in [0,u] \\ \quad \text{and } x_2 \in [0,v] \end{cases} \quad .$

3. A PNP transistor symbol is described as a complex primitive set (see Fig. 2):

$$<\text{Trans},\ f(\underline{x}),\ (0,0),\ (0,0),\ ((\text{Base},\ (0,0),\ (-6,0)),\ (\text{Collector},\ (0,0),\ (1,6)),\ (\text{Emitter},\ (0,0),\ (1,-6)))> \quad .$$

It is most convenient to describe Trans (and thus, implicitly $f(\underline{x})$) by a PDL expression:

$\text{Trans} := (\alpha(0,-4) \oplus M[4]\, c)$

$\otimes\ (\alpha(1,3) \oplus (V(3) \otimes L(-3,-2)))$ $(1,3)$

$\otimes\ (\alpha(1,-3) \oplus (\sim V(3) \otimes Ar(-3,2)))$ $(1,-3)$

$\otimes\ (\alpha(-2,0) \oplus (L(4,0) \otimes V(2) \otimes \sim V(2)))\#$ $(-2,0)$

The picture elements used in the Trans description have been defined in previous pages with the exception of $Ar(x,y)$.

$$Ar(x,y) := R[\theta]\ (M[s]\ \ell \oplus ar) \circledast Q(x,y)\#$$

The following PDL expression describes the attachment of $S_1$ to the collector, $S_2$ to the emitter, and $S_3$ to the base of the transistor:

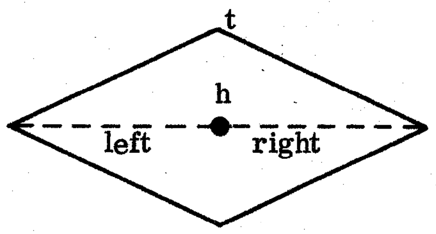
$\text{Trans} \oplus (/\text{Collector} \oplus S_1) \otimes (/\text{Emitter} \oplus S_2) \otimes (/\text{Base} \oplus S_3)$

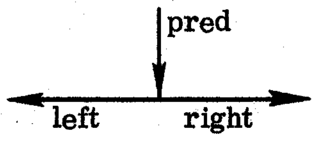
4. We will specify the predicate box of our flow chart as a complex primitive:

$<\text{pred},\ f(\underline{x}),\ (0,2),\ (0,0),\ ((\text{left},\ (0,0),\ (-3,0)),\ (\text{right},\ (0,0),\ (3,0)))>$ ,

where $f(\underline{x})$ is implicitly described by the PDL expression:

$$\alpha(0,2) \oplus ((L(-3,-2) \oplus L(3,-2)) \circledast (L(3,-2) \oplus L(-3,-2)))$$





5. A conventional alphabet of letters and special characters can be considered a complex primitive set:

$$<\text{Char}(i),\ f(\underline{x};i),\ (0,0),\ (t_1(i),0), \Lambda> ,$$

where i is restricted to the positive integers.

The alphabet is mapped into the positive integers and $f(\underline{x};i)$ is the normal form pattern for the ith character; for example $f(\underline{x};5)$

might be an upper case "E". The details of each character need not be specified here. For our purposes, it will be assumed that a rectangle bounds each character, and that the tail and head of a character are defined by the endpoints of the base line of the bounding rectangle. Furthermore, the base line is such that a string designating a word may be described as $\mathrm{Char}(i_1) \oplus \mathrm{Char}(i_2) \oplus \dots \oplus \mathrm{Char}(i_k)$ . $t_1(i) = 1$ for most i. Finally, we abbreviate the above by enclosing strings with quotation marks so that, for example, the expression "THE" is equivalent to $\mathrm{Char}(i_T) \oplus \mathrm{Char}(i_H) \oplus \mathrm{Char}(i_E)$, where $\mathrm{Char}(i_T)$ designates a "T," etc. More complicated definitions permitting the simple description of superscripts and subscripts could be given by specifying multiple attachment points.

The "No" and "Yes" extensions used in conjunction with a predicate box of a flow chart can now be described:

$$\mathrm{bv} := \mathrm{M}[0.1]\,\mathrm{R}[0.5]\,\mathrm{b}\,\#$$

$$\mathrm{No} := \ell \oplus ((\mathrm{bv} \oplus \text{"N"} \oplus {\sim}\mathrm{bv}) \circledast \ell) \oplus \ell\,\#$$

$$\mathrm{Yes} := \ell \oplus ((\mathrm{bv} \oplus \text{"Y"} \oplus {\sim}\mathrm{bv}) \circledast \ell) \oplus \ell\,\#$$

$$t \overset{N}{\overline{\qquad\quad}} h \qquad\qquad t \overset{Y}{\overline{\qquad\quad}} h$$

## D. Attribute Assignment and Display

Following Feldman and Rovner (1969), we specify attributes with associative triples of the form: "attribute of object is value" using the notation: $A.O \equiv V$, where A denotes the attribute, O the object, and V the value.

Examples

1. The resistor (Res) could be given the attributes of resistance value and tolerance. The statements:

R1 := Res #

Ohms.R1 ≡ 15000 #

Tol.R1 ≡ 10 #

would describe a resistor R1 with resistance 15000Ω and tolerance 10 percent.

2. In the flow chart system, we might either "draw" the text inside each box or associate the text as an attribute of that box. The contents of the initialization box of Fig. 1 could be described:

Initial := func #

Text.Initial ≡ "SUM := 0" #

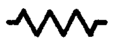
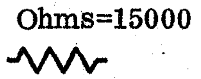
There are at least two areas of the display where it would be useful to show attributes — one is in the vicinity of the picture object with which the attribute is associated; the other is in a "working" area of the display where user commands and responses are continuously shown. The following command will be used to display a value V corresponding to an attribute/object pair:

Showattr (A.O, what, where) # ,

where A.O is the attribute/object pair, what is used to indicate whether A or O is also to be shown, and where specifies the location of the first character to be displayed. If what = A, then the attribute name is displayed; if what = O, then the object name is also displayed. A, O, and V are considered as horizontal text for display purposes. where = W indicates that the information should appear in the working area of the display; where = $\alpha(X, Y)$ specifies that the display start at the virtual picture space coordinates (X, Y) relative to the tail of the object O which is already displayed.

Examples

1. Showattr (Ohms.R1, A,$\alpha$(0,2))# will change the display

 to 

(provided R1 has been "evaluated" previously).

2. Showattr (Text.Initial, N,$\alpha$(-1.5,-1.5))# will change the display $\boxed{\phantom{\text{SUM := 0}}}$ to $\boxed{\text{SUM := 0}}$ .

3. Showattr (Tol.R1, O,W)# results in the appearance in display working area of: Tol.R1 ≡ 10 .

The argument all could be used in place of A or O to cause the display of a large number of values. For example, Ohms.all as the A.O pair in a Showattr call causes the display of values of all objects with the attribute Ohms; all.R1 results in the display of all attributes of R1. Similarly, if there is more than one use of R1 in a picture, Ohms.R1 will refer to all of these.

There are a number of implicit attributes that are automatically available from the system through the Showattr command. These include (a) the attributes Tail and Head which refer to the virtual space tail and head coordinates of the object, and (b) the attribute Label which denotes a unique label generated by the system for each primitive and higher level structure used.

The command Clearattr (A.O, where)# is employed to clear the display of the specified attribute.

## E. Interactive Drawing

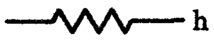
In this section, we outline the mechanics of incrementally generating and modifying pictures. It is assumed that a text-editing system which permits a user to conveniently edit input descriptions and commands is available.

The description of the current picture which has been "evaluated" and displayed is assigned the name $\sigma$. The current picture can be translated, magnified, or rotated by evaluating the expressions $(\alpha(x,y) \oplus \sigma)$, $M[s]\sigma$, and $R[\theta]\ \sigma$ respectively. When the display has been cleared, $\sigma$ describes an empty picture with tail and head at the origin. A subsequent call <u>Evaluate</u> $(S_1)\#$ is equivalent to <u>Evaluate</u> $(\sigma \oplus S_1)\#$. This first results in the evaluation of the PDL expression $S_1$ into a virtual space picture $f_1$ with tail at the origin (which is the head of the current $\sigma$); $f_1$ is then mapped according to a <u>Virtualspace</u> function into a real space picture $f_1$ which is displayed. To concatenate a picture described by $S_2$ onto a tail or head of $f_1$, either <u>Evaluate</u> $(\sigma \emptyset S_2)\#$ or <u>Evaluate</u> $(\emptyset S_2)\#$, where $\emptyset \in \{\oplus , \otimes , \ominus , \circledast\}$, may be employed. The current picture can be assigned a name n by the command:

$$n := \sigma\#$$

<u>Example</u>

<u>Evaluate</u> $(\ell \oplus \text{Res} \oplus \ell)\#$ displays:

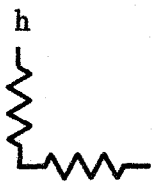
(a) 

Following this by:

$$r := \sigma\#$$

<u>Evaluate</u> $(\otimes\ R[0.5]\ r)\#$

causes the display to change to:

(b) 

The head position h is normally always indicated on the display by "h."

The head position "h" can be moved to any node in the picture in two ways. The superposition operator / can be explicitly used in an Evaluate command or the commands Movehdf, Movehdb, and Stopmove may be employed. Execution of Movehdf and Movehdb will cause the head to systematically trace "forward" or "backward" through the nodes of the picture in real time until the command Stopmove is issued. For example, the head in (b) above can be moved to the same node as (a) by

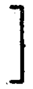
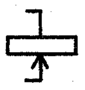
(1) Evaluate $(/(\sim R[0.5]\ r \oplus r))\#$ or

(2) by issuing Movehdb, waiting until the head appears in the desired location, and then issuing a Stopmove.

Elements of a picture may be erased by the command Delete (S)# . All parts of the current picture which locally can be described by the PDL expression are deleted.

Examples

1. Delete $(M[s]\ R[0.25]\ \ell)\#$ would remove all $45^{\circ}$ lines from the current picture.

2. Erasure of one particular pattern when many of the same class appear in the picture requires the use of label designators. Insertion of a function box in the rightmost vertical line of Fig. 1 to change  to  involves the following command sequence:

   (a) Showattr (Label.V(s), A,$\alpha$(1, 1)#

   (b) Delete (V_i)#

   (c) Movehdb #

   (d) Stopmove #

   (e) Evaluate $(\oplus\ (V(4) \oplus R[0.5]\ ar \oplus \sim func \oplus V(s_1) \oplus /M[s_2]R[1]\ell))\#$

Command (a) will display labels of all vertical lines V(s). We assume that "i" is the label for the vertical line of interest. V_i is deleted in (b). (c) and (d) move the head to the former tail position of V_i and (e) makes the desired insertion.

3. Delete $(\sigma)$ # clears the display. $\sigma$ then describes an empty picture.

Deletions and complex insertions can also be accomplished by using the text-editing system to modify the PDL expression describing the current picture.

Real-time transformations of the visual display may be performed by defining the appropriate functions over pictures using a general on-line computational facility. We may zoom in on picture parts and roll the picture horizontally or vertically across the display by specifying appropriate real-time changes in the Virtualspace function.

Examples

1. To move the current picture vertically through the display, we might use the following Algol-like procedure:

```
procedure Rollup;
    for i := 1 step 1 until n do Virtualspace (vs1-c, vs2-c)#
```

Here $\underline{c} = (0,k)$, where k is a real constant, and vs1, vs2 are the values of the current Virtualspace function parameters (assumed available to the system).

The commands:

```
Virtualspace (u, v)#
Evaluate (S)#
Rollup #
```

causes the display of that part of the picture described by S that is bounded by $(\underline{u}, \underline{v})$ and then rolls the displayed part up by changing the rectangle to $(\underline{u} - \underline{c}i, \underline{v} - \underline{c}i)$, $i = 1, \ldots, n$.

2. A picture may be magnified or "zoomed" about its head by means of the following procedure:

```
procedure zoom;
begin
    $\underline{x}$ := head ($\sigma$);
    for i := n step -1 until 1 do Virtualspace ($\underline{x} - \underline{k}i$, $\underline{x} + \underline{k}i$)
end zoom#
```

$\underline{k} = (k,k)$, k a constant.

It might be necessary to include a function Delay (t) in the loop in both Rollup and zoom; Delay (t) effectively puts the process in a wait state for t ms.

## F. The Definition and Satisfaction of Constraints

By picture constraints, we mean a set of relations over picture parts. Examples of some useful constraints are:

1. one line segment is to be parallel, perpendicular, or, in general, make any given angle with another;
2. the area of one closed figure is to be x times that of another;
3. two figures are to be connected together at a given set of attachment points.

It is often most natural to specify a picture in terms of the constraints the parts must satisfy; the exact geometry of the picture should then be automatically computed by the system. Constraints will be classified as either topological or computational depending on whether they can be defined using PDL expressions or whether they require a more general computation.

The PDL notation expresses the relation of picture concatenation. This constraint is satisfied in general by translation, rotation, and scale change. We use the arguments of the M and R transformations as free variables selected to satisfy the concatenations involving the $\circledast$ operator.

Examples

1. Let $s_1$, $\theta_1$, $s_2$, and $\theta_2$ be constants and $s, \theta$ be variables. Then, evaluation of the expression: $(R[\theta_1]M[s_1]\ell \oplus R[\theta_2]M[s_2]\ell) \circledast R[\theta]\,M[s]\,\ell$ would include the assignment of values to s and $\theta$ to satisfy: $\text{tail}\,(R[\theta]\,M[s]\,\ell) = \text{tail}\,(R[\theta_1]M[s_1]\ell) = \underline{t}$ and $\text{head}\,(R[\theta]\,M[s]\,\ell) = \text{head}\,(R[\theta_2]M[s_2]\ell) = \underline{h}$.

   We directly use the definitions of the primitives to solve for $\theta$ and s:

   $$\underline{t} = s((0,0) + \underline{c})\,A$$

   $$\underline{h} = s((1,0) + \underline{c})\,A$$

   where $A = \begin{bmatrix} \cos\theta\pi & \sin\theta\pi \\ -\sin\theta\pi & \cos\theta\pi \end{bmatrix}$

   yielding the solution $s = |\underline{h} - \underline{t}|$ and $\theta\pi = \tan^{-1}(h_1 - t_1,\ h_2 - t_2)$.

   Using the primitive L(x, y) instead of $R[\theta]\,M[s]\,\ell$ leads to a simpler set of equations and immediate solution:

   $$\underline{t} = (0,0) + \underline{c}$$

   $$\underline{h} = (0,0) + (x,y) + \underline{c}$$

2. Blank primitives are useful for describing relations. Two line segments separated by a distance s may be constantly maintained in parallel by the description:

   $$R[\theta]\,(M[s_1]\ell \otimes M[s]\,R[0.5]\,b \oplus M[s_2]\ell)$$

3. The arrow Ar(x, y) was described in section V.C as

$$Ar(x,y) := R[\theta]\,(M[s]\ell \oplus ar) \circledast Q(x,y)\#$$

The equations relating the free variables $\theta$ and s to the known tail $\underline{t}$ and head $\underline{h}$ when Ar(x, y) is evaluated for a given x and y are:

$$\underline{t} = s((0,0) + \underline{c}_1)\,A$$

$$\underline{h}_\ell = s((1,0) + \underline{c}_1)\,A$$

$$\underline{t}_{ar} = \underline{h}_\ell$$

$$\underline{t}_{ar} = ((0,0) + \underline{c}_2)\,A$$

$$\underline{h} = ((1,0) + \underline{c}_2)\,A \quad ,$$

where A is the same rotation matrix as in 1. The solution is $s = |\underline{h}-\underline{t}| - 1 = \sqrt{x^2 + y^2} - 1$ and $\theta\pi = \tan^{-1}(x, y)$.

More complicated expressions are also allowed. Our intent is that the system automatically derive the equations relating the known and unknown tails and heads, and analytically solve for the unknown variables; expressions where zero or greater than one solution exist are assumed to be in error. Similarly, when $\circledast$ appears in an expression containing no free variables, the system will verify that the appropriate points coincide.

The on-line computational facility handles non-topological constraints.

Examples

1. Suppose we wish to draw a line segment concatenated onto the head of a segment $\ell$_i and at an angle of $\theta\pi$ radians to it; assume that the head is correctly located at head ($\ell$_i). The following sequence will will perform the required generation:

   Showattr (Angle.$\ell$_i, N, W)#

   A := $\beta + \theta$ #

```
        Evaluate ($\oplus$ R[a] $\ell$) #
```

where $\beta$ is the displayed angle of $\ell$_i.

2. Let $S_1$ and $S_2$ each describe polygonal figures $f_1$ and $f_2$ respectively. Suppose we wish to magnify $f_2$ so that the area enclosed by $f_2$ is u times that enclosed by $f_1$. The following procedure will compute the area of such a closed figure:

```
    real procedure Area (t);
    begin
        start := t;
        h := $\Omega$ ;   A := 0;
        while h $\neq$ start do
        begin
            Lfp (R[$\theta$] M[s]$\ell$, t, h);
            A := A + $h_1 t_2$ - $t_1 h_2$;
            t := h
        end
        Area := abs(A)/2
    end Area #
```

where Lfp(a, b, c) is a primitive recognition driver that searches the picture (actually, some internal description of the picture) for a member of the primitive set $\mathscr{P}$(a) with tail at b and, if successful, returns the coordinates of the head of the primitive in c. $\Omega$ denotes undefined.

The sequence:

```
    s := sqrt (u $\otimes$ Area($t_1$)/Area ($t_2$)) #
    n := M[s] $S_2$ #
```

will then assign to the name n a description of the magnified figure whose area is u times that of $S_1$; $\underline{t}_1$ and $\underline{t}_2$ are the tails of $S_1$ and $S_2$.

We expect a library of routines such as Area to be available to a user so that the procedures for the most common computational constraints need not be defined at every use.

## G. Flow Chart and Electric Circuit Generation

The primitives and structures presented in earlier pages are employed to generate the flow chart of Fig. 1:

$S := R[1]c \oplus Ar(0,-3) \oplus func \oplus L(0,-3) \oplus ((R[-0.5]\, ar \oplus func \oplus Ar(0,-3) \oplus pred) \circledast (\sim(/right \oplus No \oplus Ar(0,-3) \oplus func \oplus L(0,-2) \oplus L(5,0) \oplus L(0,y) \oplus Ar(x,y)))) \oplus /left \oplus \sim Yes \oplus Ar(0,-3) \oplus func \oplus Ar(0,-3) \oplus R[1]\, c$ #

Delete ($\sigma$) #

Virtualspace ((-20,-50), (20,10)) #

Evaluate (S) #

Text.pred ≡ "X=EOF" #

Showattr (Label.func, O, W) #

Text.func_1 ≡ "SUM :=0" #

Text.func_2 ≡ "READ X" #

Text.func_3 ≡ "SUM := SUM + X" #

Text.func_4 ≡ "WRITE X" #

Showattr (Label.c, O, W) #

Text.c_1 ≡ "START" #

Text.c_2 ≡ "HALT" #

Showattr (Text.all, N, $\alpha(-1.5,-1.5)$) #

Clearattr (Label.all, $\alpha(x,y)$) #

The sequence above assumes that the first <u>Showattr</u> indicates labels of 1, 2, 3, and 4 for the four "func"'s and labels of 1 and 2 for the two "c"'s.

If the flow charts were being designed interactively, the entire expression S would be normally created and evaluated incrementally with appropriate deletions, additions, and commands to move the head forward and backward. The textual contents of each box could be given as part of the flow chart PDL expression rather than as attributes. The above flow chart construction method should be compared with the on-line flow chart language FPL/I of Richardson (1968) which uses a lightpen and symbol lightbuttons for drawing.

The series/parallel resistance network of Fig. 2 can be generated in a similar manner. We will describe the picture part enclosed by the dotted lines:

$\mathrm{Rh}(s1,\ s2) := M[s1]\ell \oplus \mathrm{Res} \oplus M[s2]\ell$ #

$S := H(2) \oplus ((((V(3) \oplus \mathrm{Rh}(1, 0.5) \oplus \mathrm{Rh}(1, 1) \oplus {\sim}V(3)) \circledast \mathrm{Rh}(3, s)) \oplus{\sim}V(1))$

$\circledast ({\sim}V(3) \oplus ((\mathrm{Rh}(1, 2) \oplus{\sim}V(1)) \circledast ({\sim}V(3) \oplus \mathrm{Rh}(1, 2) \oplus V(2)))$

$\oplus \mathrm{Rh}(1, 1) \oplus V(3))) \oplus H(3)$ #

The resistance values may be assigned and displayed for each resistor as described earlier. Again, because of the complexity of the entire description S, it would be normal to incrementally produce and display S.

## VI. PICTURE INTERPRETATION

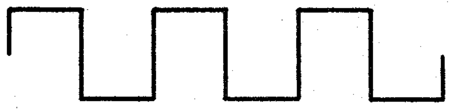
### A. Picture Grammars and Syntax Analysis

The set of well-formed pictures for a given application is specified by a context-free grammar G generating a language $L(G) \subseteq \mathrm{PDL}$. G describes the pictures:

$$\mathscr{P}_G = \bigcup_{S \in L(G)} \mathscr{P}(S) \quad .$$

G also serves the function of imposing a higher level structure or hierarchic description H on any picture f in $\mathscr{P}_G$. This structure is determined by the derivation of the description S of f according to G (ignoring questions of ambiguity) and is conveniently represented as a tree. G will consist of a 4-tuple: $G = (V_N, V_T, P, D)$, where $V_N$ is a set of non-terminal or intermediate symbols; $V_T$ is the set of terminal symbols of the PDL syntax; P is a set of rules of the form: $A \rightarrow pd11 | pd12 | --- | pd1n$, where $A \in V_N$, $n \geq 1$, and each pd1i is a PDL expression with the addition that non-terminal symbols may appear in place of primitive names; and D is a distinguished symbol in $V_N$ from which all sentences of L(G) are generated.

We illustrate these ideas by the following simple example which will generate descriptions of square wave trains of the form:



$G = (V_N, V_T, P, \text{TRAIN})$

$V_N = \{\text{TRAIN, SQUARES, CYCLE, TOP, BOT}\}$

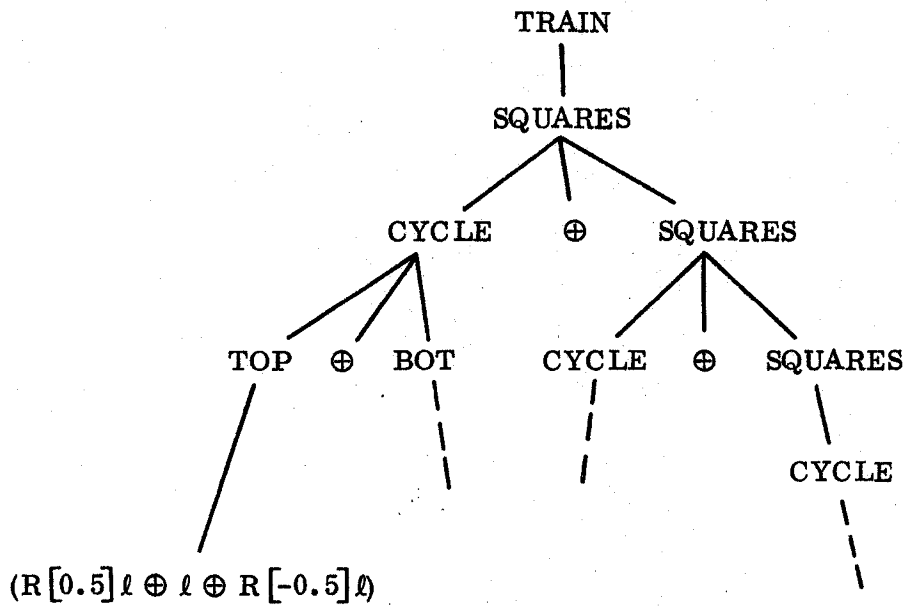
$V_T = \{\oplus, R, [,], \ell, 0.5, -0.5, 1, (,)\}$

$$\begin{aligned} P = \{\text{TRAIN} &\rightarrow \text{SQUARES}, \\ \text{SQUARES} &\rightarrow \text{CYCLE} \mid \text{CYCLE} \oplus \text{SQUARES}, \\ \text{CYCLE} &\rightarrow \text{TOP} \oplus \text{BOT}, \\ \text{TOP} &\rightarrow (R[0.5]\ell \oplus \ell \oplus R[-0.5]\ell), \\ \text{BOT} &\rightarrow \sim R[1]\text{TOP}\} \end{aligned}$$

A description of the above wave train is derived by applying rules of P starting with TRAIN:

$$\text{TRAIN} \Longrightarrow \text{SQUARES} \Longrightarrow \text{CYCLE} \oplus \text{SQUARES}$$
$$\Longrightarrow \text{CYCLE} \oplus \text{CYCLE} \oplus \text{SQUARES} \Longrightarrow \text{CYCLE} \oplus \text{CYCLE} \oplus \text{CYCLE}$$
$$\Longrightarrow \text{TOP} \oplus \text{BOT} \oplus \text{CYCLE} \oplus \text{CYCLE}$$
$$\Longrightarrow (\text{R}[0.5]\ell \oplus \ell \oplus \text{R}[-0.5]\ell) \oplus \text{BOT} \oplus \text{CYCLE} \oplus \text{CYCLE}$$
$$\Longrightarrow (\text{R}[0.5]\ell \oplus \ell \oplus \text{R}[-0.5]\ell) \oplus \sim\text{R}[1]\ \text{TOP} \oplus \text{CYCLE} \oplus \text{CYCLE}$$
$$\text{---} \Longrightarrow (\text{R}[0.5]\ell \oplus \ell \oplus \text{R}[-0.5]\ell) \oplus \sim\text{R}[1]\ (\text{R}[0.5]\ell \oplus \text{--})$$
$$\text{---} \oplus \sim\text{R}[1]\ (\text{R}[0.5]\ell \oplus \ell \oplus \text{R}[-0.5]\ell)$$

This derivation gives a hierarchic structure H which may be represented by the tree (partially drawn):



Given a grammar G, a picture f is <u>syntactically</u> analyzed by <u>parsing</u> it according to G to obtain its PDL description S and hierarchic description H. We accomplish this by means of a general picture syntax analyzer which accepts a grammar and uses the latter to drive a set of primitive recognizers around a given picture (its internal representation). Note that the picture is analyzed rather than the PDL expression used to generate the picture. This is because we expect the descriptions specified by a user when generating pictures to be different from those desired in an analysis; in fact, we might want to analyze the same picture according to several different grammars (corresponding to different syntactical interpretations).

A grammar is specified to the system by the command:

<u>Grammar</u> n;

$P = \{r_1, r_2, ---, r_n\}$ #

where n is the name associated with the grammar and each $r_i$ is a rule. The left part of $r_1$ is taken as the distinguished symbol. A picture is analyzed according to a given grammar by the command:

<u>Parse</u> (n, S, G) #

where S is a picture description (or name assigned to a description), n is a name assigned to the results of the parse, and G is the name of a grammar. <u>Parse</u> (n, $\sigma$, G)# will analyze the currently displayed picture. Parse (n, S, G) # will first evaluate S to a picture if $S \neq \sigma$ and display the results.

As each primitive is recognized during a parse, it is eliminated from the picture. An abstracted version of the parse as well as the location of the last primitive found is continually displayed alternating with the residue picture (the original minus the eliminated primitives) as described in Shaw (1968, 1969b); our abstracted picture consists of its "graph" with tail and head of each edge geometrically located at the coordinates of the primitive it describes. This is useful for debugging grammars and for finding the points of non-well-formedness in a picture.

A difficult but unresolved problem at this point is how to present the results of the parse. The large tree generated by the extremely simple grammar and picture above illustrates the nature of our problem. After indicating that a picture is well-formed (a successful parse), it would be desirable to present selected portions of the tree in pictorial form. Trees may be described in PDL and it should be feasible to derive a general tree display program within our system.

## B. Flow Chart and Electric Circuit Grammars

Grammars are presented for conventional flow charts and for series/parallel resistance networks. The grammar for each picture class is interesting in the sense that any valid PDL expression generated by it will describe only pictures of the class (or no picture) and any picture of the class may be parsed according to the grammar to yield a useful hierarchic description.

For flow charts, we use the primitives $\ell, \lambda, c$, and pred; in addition, the structures func, ar, Yes, and No are treated as primitives. A well-formed flow chart (Fig. 1) will have the following properties:

1. It will contain one start box and at least one finish box.
2. A "program" path exists from the start box to every other box.
3. Any number of lines may enter a function or predicate box; one line always exits from a function box and two lines from a predicate box.
4. The orientation of the line segment formed by the tail and head of every primitive will be restricted to one of the four horizontal and vertical directions: $\rightarrow, \leftarrow, \uparrow, \downarrow$, with the exceptions of func which is restricted to $\uparrow$ and $\downarrow$, and c which we maintain as $\downarrow$.
5. Within the above restrictions, there are no limitations on the chart connectivity; the flow chart need not form a planar graph.

The rules of the flow chart grammar are:

| | |
|---|---|
| FC | $\longrightarrow \text{Start} \oplus (\text{S} \otimes /(\text{S} \oplus \text{Finish}))$ |
| S | $\longrightarrow \text{Seq} \oplus \text{Xfer} \mid \text{Xfer}$ |
| Seq | $\longrightarrow \text{FUNC} \oplus \text{Nar} \mid \text{FUNC} \oplus \text{Nar} \oplus \text{Seq}$ |
| Xfer | $\longrightarrow \text{Cond} \mid \text{FUNC} \oplus \text{Gar} \mid \text{Finish}$ |
| Cond | $\longrightarrow \text{PRED} \oplus \text{Br} \mid \text{Cond\_i}$ |
| Br | $\longrightarrow ((/\text{left} \oplus \sim\text{No} \oplus \text{S1}) \otimes (/\text{right} \oplus \text{Yes} \oplus \text{S1})) \mid$ |
| | $((/\text{left} \oplus \sim\text{Yes} \oplus \text{S1}) \otimes (/\text{right} \oplus \text{No} \oplus \text{S1}))$ |
| S1 | $\longrightarrow \text{Gar} \mid \text{Nar} \oplus \text{S}$ |
| Nar | $\longrightarrow \text{L} \oplus \lambda\_\text{g} \oplus \text{AR} \mid \text{Nar\_i}$ |
| Gar | $\longrightarrow \text{L} \oplus \text{AR} \oplus /\lambda\_\text{g} \mid \text{Gar\_i}$ |
| L | $\longrightarrow \text{LH} \mid \text{LV}$ |
| LH | $\longrightarrow \text{H} \mid \text{H} \oplus \text{V} \mid \text{H} \oplus \text{V} \oplus \text{LH}$ |
| LV | $\longrightarrow \text{V} \mid \text{V} \oplus \text{H} \mid \text{V} \oplus \text{H} \oplus \text{LV}$ |
| H | $\longrightarrow \text{M}[\text{s}]\,\ell \mid \sim\text{H}$ |
| V | $\longrightarrow \text{M}[\text{s}]\,\text{R}[0.5]\,\ell \mid \sim\text{V}$ |
| AR | $\longrightarrow \text{ar} \mid \text{M}[0.5]\,\text{ar} \mid \sim\text{AR}$ |
| Start | $\longrightarrow \text{C} \oplus \text{Nar}$ |
| Finish | $\longrightarrow \text{C\_f} \mid \text{Finish\_i}$ |
| C | $\longrightarrow \text{R}[1]\,\text{c}$ |
| FUNC | $\longrightarrow \text{func} \mid \sim\text{func}$ |
| PRED | $\longrightarrow \text{pred} \mid \text{R}[1]\,\text{pred} \mid \text{R}[0.5]\,\text{pred} \mid \text{R}[-0.5]\,\text{pred}$ |

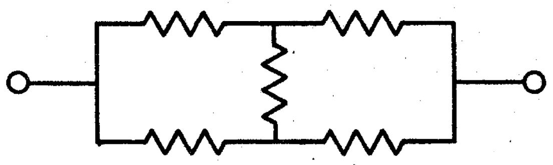
The right parts $\text{M}[\text{s}]\,\ell$ and $\text{M}[\text{s}]\,\text{R}[0.5]\,\ell$ designate the set of all horizontal and vertical lines respectively.

The complexity of the above grammar and extensive use of labels is due to the generality of acceptable flow charts and the complete detail contained in generated descriptions; any reasonable flow chart is described. The major intermediate symbols have the meaning:

| | |
|---|---|
| Seq | : Sequential statements |
| Xfer | : Transfer statements |
| Nar | : Normal arrow |
| Gar | : GoTo arrow |
| Cond | : Conditional statement |
| Br | : The 2 exits of a conditional statement |
| L | : A sequence of concatenated line segments |

The labels and retracing are used to ensure at least one finish box, to allow arrows to go to any box, and to distinguish between predicate boxes.

The network grammar is considerably simpler. We will use c, ℓ, and Res as primitives and allow only horizontal and vertical orientations as before. Well-formed circuits must be properly nested; e.g.,

 is not permitted.

The grammar rules are:

1. $\text{SPRN} \rightarrow \text{Term} \oplus \text{CCT} \oplus \text{Term}$
2. $\text{CCT}_1 \rightarrow \text{Basic} \oplus \text{L} \mid \text{Basic} \oplus \text{CCT}_2$
3. $\text{Basic} \rightarrow \text{L} \oplus \text{RES} \mid \text{L} \oplus \text{Par}$
4. $\text{Par}_1 \rightarrow (\text{CCT}_1 \circledast \text{CCT}_2) \mid (\text{Par}_2 \circledast \text{CCT})$
5. $\text{RES}_1 \rightarrow \text{Res} \mid \text{R}\,[0.5]\,\text{Res} \mid {\sim}\text{RES}_2$

6. L $\rightarrow$ LH | LV | $\lambda$

7. LH $\rightarrow$ H | H $\oplus$ V | H $\oplus$ V $\oplus$ LH

8. LV $\rightarrow$ V | V $\oplus$ H | V $\oplus$ H $\oplus$ LV

9. Term $\rightarrow$ M[0.1] C

10. C $\rightarrow$ c | R[0.5] c | ~C

(The subscripts on some of the intermediate symbols are referred to in the next section and can be ignored at present.)

C. Semantic Interpretation

We now discuss how pictures may be further interpreted by employing the results of a syntax analysis or by working with the picture directly. A particularly appealing approach is to impose a set I of interpretation rules in 1-1 correspondence with the rules of the grammar (Anderson, 1968; Evans, 1968; George, 1969), in a similar manner as in some programming language translator writing systems (Feldman and Gries, 1968). Each time a reduction is made during a parse, the corresponding interpretation rule is executed. A rule of I may consist of any sequence of computations.

For example, an equivalent single resistor circuit can be obtained from an arbitrary SPRN by executing the following interpretation rules during a parse. Rule i corresponds to syntactic rule i of the grammer in the last section:

1. S := M[0.1] R[-0.5] c $\oplus$ $\ell$ $\oplus$ Res $\oplus$ $\ell$ $\oplus$ M[0.1] R[-0.5] c;
   OHMS.Res $\equiv$ v(CCT);

2. v($CCT_1$) := v(Basic); | v($CCT_1$) := v(Basic) + v($CCT_2$);

3. v(Basic) := v(RES); | v(Basic) := v(Par);

4. v($Par_1$) := (v($CCT_1$) $\times$ v($CCT_2$))/(v($CCT_1$) + v($CCT_2$)); |
   v($Par_1$) := (v($Par_2$) $\times$ v(CCT))/(v($Par_2$) + v(CCT));

5. $v(RES_1)$ := OHMS.Res; | $v(RES_1)$ := OHMS.Res; | $v(RES_1)$ := $v(RES_2)$;

6. $\Lambda$

7. $\Lambda$

8. $\Lambda$

9. $\Lambda$

10. $\Lambda$

$\Lambda$ denotes an empty rule. v(x) means "the value of the structure generated by x." The vertical bar "|" separates alternate right parts of a rule. We use subscripts, e.g., $CCT_1$, and $CCT_2$, to distinguish between identically named elements. Rules 2 and 4 contain the basic computations for equivalent resistance in series and parallel networks respectively. Rule 1 generates a description of the equivalent circuit and is the last rule executed during a successful parse.

The command: Parseandinterpret (S, G, I)# will parse the picture described by S according to the grammar G, and, at the same time, will apply the rules of I to interpret f. If S has been parsed by invoking Parse (n, S, G)#, we can interpret the results in n according to I by using the command Interpret (n, I)#. Several semantic interpretations may then be made on a picture by calling Interpret more than once.

The program represented by a drawn flow chart can be executed by working directly with either the picture or the results of a parse. In the former case, we employ primitive pattern recognition routines, while in the latter, we search the hierarchic description produced by the parse. It is more convenient here to assume prior parsing since many irrelevant pictorial details may be ignored. Let Find (n, t, m) be a Boolean routine that searches the parsing tree for a structure named n with tail at t and returns a pointer to the structure in m if successful; Find will return true if successful and false otherwise. The following procedure ExecuteFC(t) will then execute the program represented by any well-formed flow chart which may be drawn; the parameter t is the tail coordinates of a flowchart box.

```
procedure ExecuteFC(t);
begin
    comment Execute computations in boxes;
    if Find (C, t, a) then
    begin if Text.a ≡ "HALT" then Return end
    else
    if Find (FUNC, t, a) then EX(Text.a)
    else
    if Find (PRED, t, a) then
    begin
        Find(R[θ] left, head(a), b);
        Find(R[θ] right, head(a), c);
        if EX(Text.a) then
        begin if Find(Yes, head(c), d) then a := d
            else Find(~Yes, head(b), a)
        end
        else begin if Find(No, head(c), d) then a := d
                else Find(~ No, head(b), a)
            end
    end
    comment Trace arrow to next box and call ExecuteFC recursively;
    if Find(Nar, head(a), b) then ExecuteFC(head(b))
    else
```

```
        begin Find(Gar, head(a),b);
              Find(AR, head(b),c);
              ExecuteFC (head(c))
        end
    end ExecuteFC #
```

EX(text) is a routine that executes the program represented by the textual string in its argument; EX will return true or false if its argument is a Boolean expression. After drawing, displaying, and assigning the appropriate textual attributes to a flow chart, the command

ExecuteFC((0, 0))#

will cause its real-time execution.

When it is not convenient or useful to interpret pictures by computations over its parse, the picture itself may be used in the computation as illustrated by the Area procedure given in Section V.F.

## VII. CONCLUDING REMARKS

The preceding pages have outlined a set of generation and interpretation facilities for a proposed general purpose computer graphics system. The usefulness, convenience and adequacy of these facilities were illustrated by examples in flow chart generation and execution, and electric circuit design. Earlier research work has demonstrated the feasibility of the more novel components of the system — the picture description scheme and picture grammars (Miller and Shaw, 1968a; Shaw, 1968, 1969a), the parsing techniques (Shaw, 1968, 1969b), and the use of string descriptions for interactive drawing (George and Miller, 1968; George, 1968). It is evident that many details remain to be worked out. A system of this type may be implemented conveniently using a translator writing system for interactive languages (e.g., George, 1969) so that changes can be easily made as the details evolve.

The major features of our system which serve to distinguish it from others are:

1. Pictures are specified in a virtual picture space by symbolic descriptions; display generation occurs by executing these descriptions and mapping the resulting picture into the real space of the display device.
2. Attributes may be computed and assigned to pictures of any complexity.
3. Picture grammars are employed to define the class of pictures for a specific application. Syntactic description and well-formedness are obtained by parsing pictures according to a given grammar.
4. Further interpretation of pictures occurs by specifying computations over the picture and over the hierarchic structure obtained from a picture parse.

All of the above features will operate in an interactive mode. Such a system should provide a simpler yet complete means for interactively generating and interpreting artificial pictures.

## ACKNOWLEDGEMENTS

The author is grateful to William F. Miller for his continued encouragement and support of this work, and to James George and Charles Zahn for several useful discussions related to this paper. The accurate and efficient typing of the draft by Kathleen Maddern and Cherie Matheson is gratefully acknowledged.

## REFERENCES

Anderson, R. H. (1968). Syntax-directed recognition of hand-printed two-dimensional mathematics. Ph.D. Thesis, Applied Math., Harvard University (January).

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

Evans, T. G. (1968). A description-controlled pattern analyzer. Proc. of IFIP Congress 68, Booklet H, pp. H152-H157.

Feldman, J., and Gries, D. (1968). Translator writing systems. Comm. ACM 11, 2 (February), pp. 77-113.

Feldman, J. A., and Rovner, P. D. (1969). An Algol-based associative language. Comm. ACM 12, 8 (August), 439-449.

George, J. E. (1968). CALGEN - an interactive picture calculus generation system. Tech. Report No. CS 114, Computer Science Department, Stanford University (December).

--- (1969). The system specification of GLAF: a linear string graphical language facility. GSG 61, Computation Group, Stanford Linear Accelerator Center, Stanford University (February).

George, J. E., and Miller, W. F. (1968). String descriptions of data for display. Report No. SLAC-PUB-383, Stanford Linear Accelerator Center, Stanford University. Presented at 9th Annual Symposium of the Society for Information Display.

Kulsrud, H. E. (1968). A general purpose graphic language. Comm. ACM 11, 4 (April), pp. 247-254.

Miller, W. F., and Shaw, A. C. (1968a). A picture calculus. Proc. Conf. on Emerging Concepts in Computer Graphics. W.A. Benjamin Press, New York.

--- (1968b). Linguistic methods in picture processing - a survey. Proc. AFIPS 1968 Fall Joint Computer Conference, Vol. 33, Thompson, Washington, D.C., pp. 279-290.

Newman, W. M. (1968). A system for interactive graphical programming. Proc. AFIPS 1968 Spring Joint Computer Conference, pp. 47-54.

Pankhurst, R. J. (1968). GULP - a compiler-compiler for verbal and graphic languages. Proc. 1968 ACM National Conference, pp. 405-421.

Richardson, F. K. (1968). Graphical specification of computation. Report No. 257, Department of Computer Science, University of Illinois (April) (Ph.D. thesis).

Roberts, L. G. (1964). Graphical communications and control languages. Second Congress on the Information System Sciences, Spartan Books, Washington, D.C., pp. 211-217.

Rosenfeld, A. (1969). Picture Processing by Computers, Academic Press, New York.

Rully, A. D. (1968). A subroutine package for FORTRAN. IBM Systems Journal 7, 3 and 4, pp. 248-256.

Shaw, A. C. (1968). The formal description and parsing of pictures. Ph.D. Thesis, Computer Science Department, Stanford University (June). Available as Report No. SLAC-84, Stanford Linear Accelerator Center, Stanford University, Stanford, California.

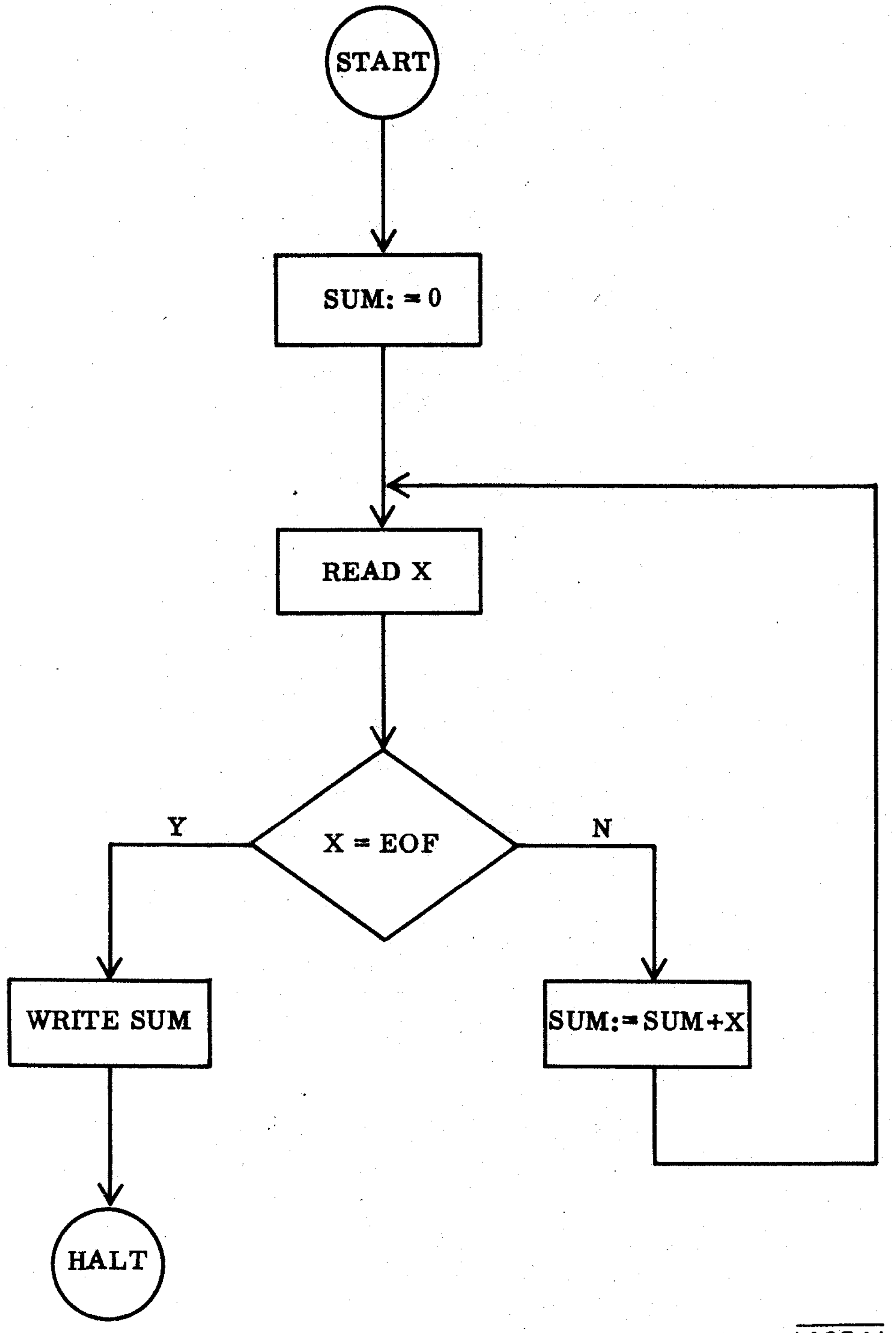
--- (1969a). A formal picture description scheme as a basis for picture processing systems. Inf. Control 14, 1 (January), pp. 9-52.

--- (1969b). Parsing of graph-representable pictures. Tech. Report No. 69-34, Department of Computer Science, Cornell University (April).

Sutherland, I. E. (1963). Sketchpad: a man-machine graphical communications system. Tech. Report No. 296, Lincoln Laboratory, Massachusetts Institute of Technology (January).

Sutherland, W. R., and Forgie, J. W. (1969). Graphics in time-sharing: a summary of the TX-2 experience. Proc. AFIPS 1969 Spring Joint Computer Conference, pp. 629-636.

## LIST OF FIGURES



1423A1

Fig. 1

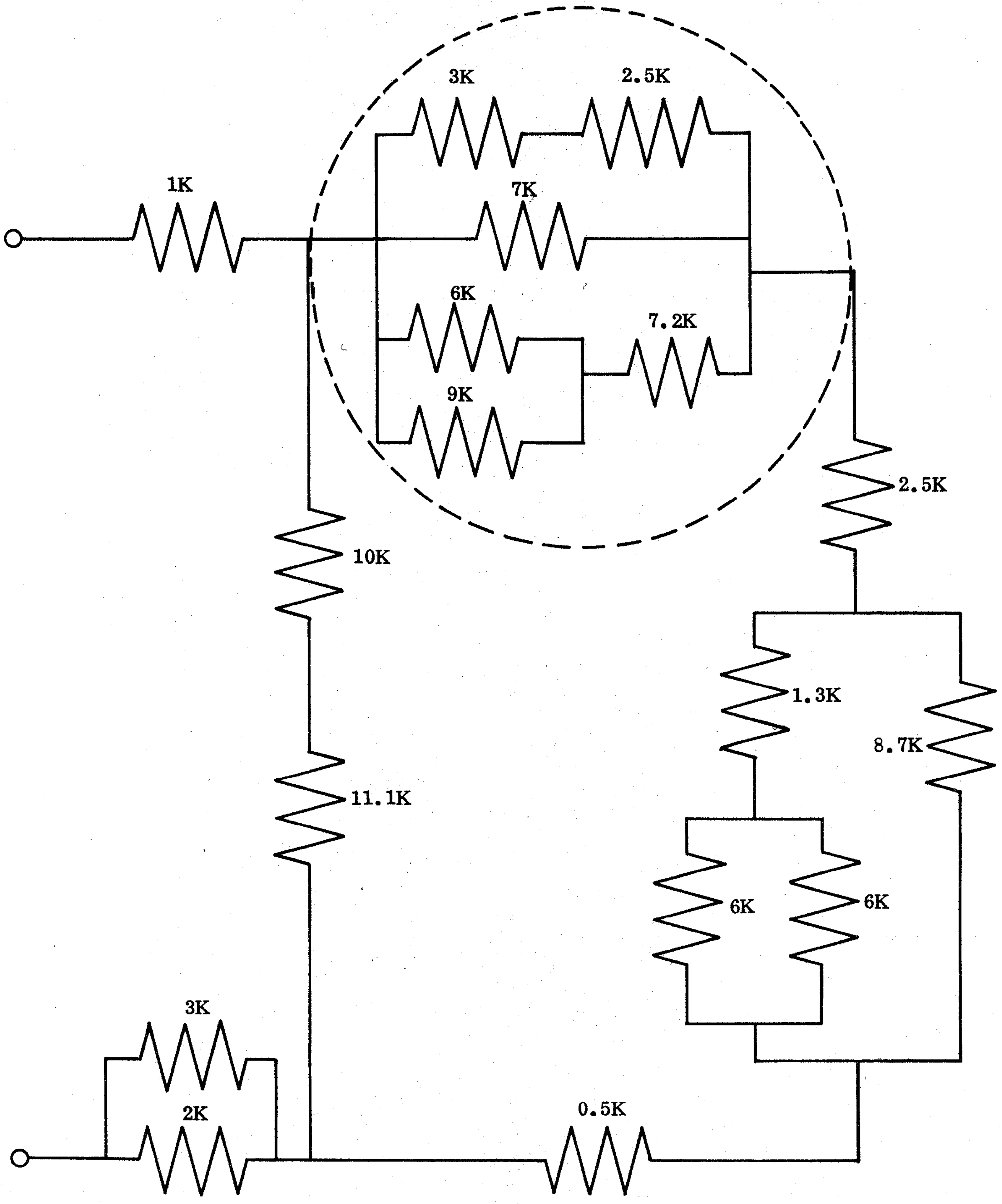
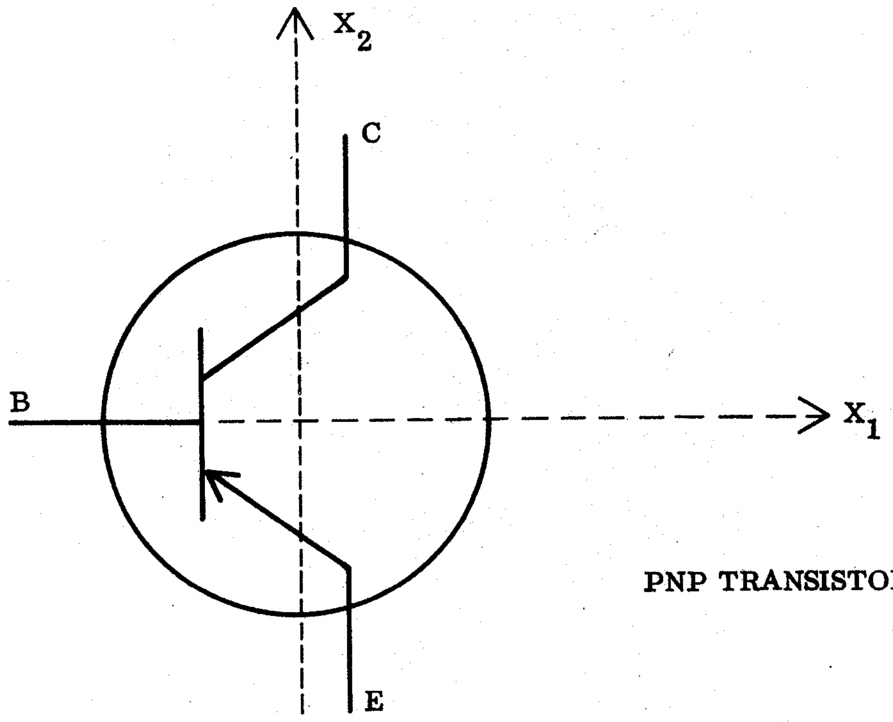
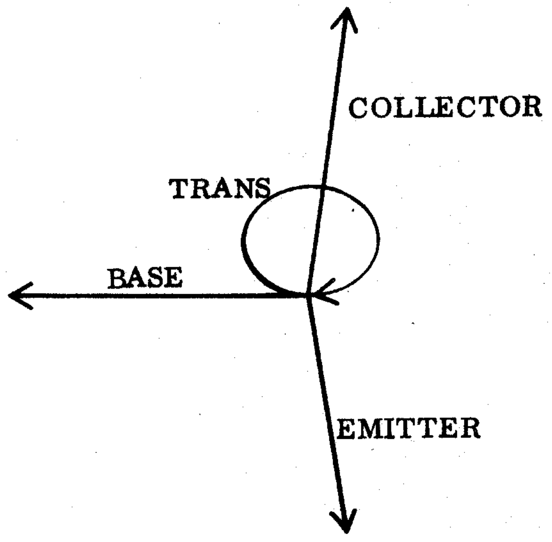


Fig. 2

1423A2



PNP TRANSISTOR



GRAPH OF TRANS

$\overline{1423A3}$

Fig. 3