PARSING OF GRAPH-REPRESENTABLE PICTURES

Alan C. Shaw

Technical Report

No. 69-34

April 1969

Department of Computer Science
Cornell University
Ithaca, New York 14850

ABSTRACT

This paper describes a syntax-directed picture analysis system based on a formal picture description scheme. The system accepts a description of a set of pictures in terms of a grammar generating strings in a picture description language; the grammar is explicitly used to direct the analysis or parse, and to control the calls on pattern classification routines for primitive picture components. Pictures are represented by directed graphs with labelled edges, where the edges denote elementary picture components and the graph connectivity mirrors the picture component connectivity; blank and don't care "patterns" allow the description of simple relations between visible patterns. The bulk of the paper is concerned with the picture parsing algorithm which is an n-dimensional analog of a classical top-down string parser, and an application of an implemented system to the analysis of spark chamber film. The potential benefits of this approach, as demonstrated by the application, include ease of implementation and modification of picture processing systems, and simplification of the pattern recognition problem by automatically taking advantage of contextual information.

CR CATEGORIES: 3.63, 4.19, 3.17

KEY PHRASES: picture processing, picture parsing, picture description, picture analysis, pattern recognition, top-down analyzer, graph representable pictures, syntax directed analysis, picture language, picture grammar, graphics language.

PARSING OF GRAPH-REPRESENTABLE PICTURES*

Alan C. Shaw
Cornell University,** Ithaca, N.Y.

INTRODUCTION

Much research and development effort has been devoted to the design and construction of hardware and computer programs for automatic picture processing. In addition to the interesting theoretical problems generated by this activity, there is a practical demand for these systems in such diverse areas as biomedicine, high energy particle physics, library automation, military reconnaissance, robot development, space exploration, computer-aided design, and commercial data processing. Several working systems exist, notably in high energy particle physics [1]; however, these are usually one-of-a-kind, represent tremendous investments, especially in programming; and, in terms of their recognition capability and efficiency, often do not perform as well as originally anticipated or desired. What is needed are some general methods that simplify the construction of automatic picture processing systems and, at the same time, improve their accuracy and efficiency.

---

* This research was supported in part by the U.S. Atomic Energy Commission, the SLAC-IBM Graphics Study Project, and the National Science Foundation grant GP-7615. Most of this work was performed at the Stanford Linear Accelerator Center, Stanford, California, and appears as part of the author's Ph.D. thesis [19]; some extensions of the thesis developed at Cornell are also described in the paper.

** Department of Computer Science,

A particularly promising approach is the linguistic one, originally advocated by Eden [4,5], Narasimhan [17,18], and Kirsch [11]. (See Miller and Shaw [14] for a survey of the use of linguistic methods in picture processing.) The method is based on the use of picture description schemes that include a mechanism for grouping components into higher level structures; a multi-dimensional extension of the techniques of language syntax analysis is then employed to analyze the pictures. Experimental systems of this type have been implemented by Narasimhan [18], Anderson [2], and Evans [6], as well as by this author.

This paper discusses the methods employed to analyze or parse pictures which may be described in our particular picture language. The applicable class of pictures is first defined; we call these graph-representable since they may be conveniently represented as a directed graph. The picture description scheme and its formal properties are described elsewhere [19,20]. Here, it is presented in sufficient detail to understand the remainder of the material. The bulk of the paper is concerned with our picture parsing algorithm — a top-down goal-directed syntax analyzer that has a direct analogy to a corresponding string syntax analyzer; we discuss the rationale behind our choice of analyzer, its advantages and limitations, the interesting differences between string and picture parsing, and the role of pattern recognition routines. Some experiences with an

implemented system are then related. The concluding section summarizes the main features of our approach and suggests some problems for future research.

The work can be distinguished from that of other researchers in the following ways:

1. It is based on a formal picture description scheme which allows the manipulation of descriptions.

2. The same description language can be used for both analysis and generation purposes.

3. It is beneficial to do the primitive pattern classification within our system rather than external to it as is done in [2] and [6]; the syntax-directed nature of the method results in a form of contextual pattern recognition that simplifies the classification task in many instances.

4. While we feel that our methods can be applied to a wide variety of pictures, and, in principle, are universally applicable, there are nevertheless many types of pictures for which our approach does not seem practical. Systems that allow the description of a picture as a set of subpictures and an arbitrary set of relations satisfied by them [6,16] are more general but, in our estimation, are not likely to be practical.

GRAPH-REPRESENTABLE PICTURES

The elementary or primitive components of a picture are informally defined as those patterns which can be conveniently treated as a unit, rather than in terms of their subparts. More specifically, a picture primitive may be any n-dimensional

pattern with two distinguished points, a tail and a head. Primitives can be geometrically concatenated together only at their tail and head points. Because only two points of possible concatenation are specified, a primitive can be represented as a labelled directed edge of a graph, pointing from its tail to head node (Figure 1).

In many applications, the absence of a specific visible pattern in a particular area of a picture is a necessary part of its description. An example is a photograph of high energy particle physics reactions; the apparent stopping of a particle track and the later appearance of several tracks emanating from the same vertex indicates the presence of an unseen neutral particle (Figure 2(a)). Blank (invisible) and "don't care" patterns connecting disjoint primitives are also extremely useful for describing simple geometric relations, such as those between adjacent characters of a word and adjacent words in text. When a relationship is to be described between two disjoint primitives separated by other patterns, the separating area is defined as a particular type of "don't care" primitive. Blank and "don't care" primitives of an arbitrary variety are therefore allowed. One special primitive, the null point primitive λ is defined; λ has an identical tail and head and consists only of the tail (head) point. λ is represented as a labelled node of a graph.

A primitive is treated as a member of a pattern class; the latter may be defined by a name, a tail and head specification,

and a Boolean function on properties which its members satisfy. We define $\mathcal{P}(x)$ to be the set of all patterns with primitive name x. A particular primitive α is described by a pair $D(\alpha) = (n,v)$, where n is the name of the pattern class to which it belongs and v is a list containing the tail and head coordinates and an arbitrary list of attributes.

Example:

Let arc be the name of the class of all two-dimensional pictures $\mathcal{P}(\text{arc})$ consisting of an arc of a circle subtending an angle of less than 180°, with tail at the clockwise extremity and head at the counterclockwise extremity of the arc. Then a picture $\alpha \in \mathcal{P}(\text{arc})$ with radius r, $\text{tail}(x_1,y_1)$, and $\text{head}(x_2,y_2)$ could be described as

$$D(\alpha) = (\text{arc},((x_1,y_1),(x_2,y_2),r))$$

Underlying the above definitions is the assumption that there exists one recognition function — a pattern recognition routine — for each primitive class. On a successful recognition, the function yields the description of the primitive.

At the primitive level, a picture can be represented as a directed graph, where the edges are the abstracted primitives labelled by their class names, some nodes may be labelled λ, and the graph connectivity mirrors the tail head concatenations of the primitives. A picture is said to be connected if, upon making each edge of its corresponding graph undirected, the resulting graph is connected. The following assumption is then made: All pictures are connected. By a straightforward use of blank and don't care primitives, a wide variety

of pictures can be represented within this framework; these include spark and bubble chamber photographs, text, flowcharts, circuits, and general line drawings. We will call these pictures graph-representable.

Example:

Figure 2(b) contains a directed graph representation of the picture in Figure 2(a). start is a blank primitive with the tail at the origin of the picture (typically, the origin of the coordinate system in which the picture is represented) and head at the beginning of the first particle track em . em is a visible track of negative curvature; ep is a visible track of positive curvature. The primitive en represents the invisible neutral particle track. The node labelled λ is explicitly distinguished to indicate the collision with a proton in this abstracted set of physics reactions (a moving negative particle interacts with a stationary positive particle producing a neutral particle which eventually decays into a positive and negative particle pair).

THE PICTURE DESCRIPTION SCHEME

The primitive components of a picture and their tail/head connectivity are described by a string in the picture description language, PDL. The following syntax will generate any sentence S in PDL (expressed S ∈ PDL).

S → p|(S$S)|(~S)|SL|(/SL)
SL → S^{ℓ}|(SL ∅ SL)|(~SL)|(/SL)
∅ → +|×|-|*
p → {any primitive class name}
ℓ → {any label designator}

Example:

The picture of Figure 2 may be described in PDL by the string: (start + ((em + λ) + (en + (em × ep)))) .

For any $S \in PDL$, we define $\mathcal{P}(S)$ as the set of all pictures with description S . At this level a picture α is described by the pair $T(\alpha) = (T_s(\alpha), T_v(\alpha))$, where $T_s(\alpha) \in PDL$ and $T_v(\alpha)$ is a list of the descriptions $D(\beta)$ of each primitive β in the picture. Not only primitives, but all pictures have a tail and a head determined by their descriptions; concatenations among pictures can only occur at their tail and head positions. Consider the picture α consisting of two subpictures α_1 and α_2 such that $\alpha_1 \in \mathcal{P}(S_1)$, $\alpha_2 \in \mathcal{P}(S_2)$ and $T_s(\alpha) = (S_1 \phi S_2)$, S_1 , $S_2 \in PDL$. Then the tail and head of α according to $T_s(\alpha)$ is defined:

$$\text{tail}(\alpha) = \text{tail}(\alpha_1)$$
$$\text{head}(\alpha) = \text{head}(\alpha_2) .$$

In the same manner as primitives, more complex pictures can thus also be represented by a directed edge of a graph.

The meaning of the binary concatenation operators $\{+,-,\times,*\}$ is presented below by defining $\mathcal{P}((S_1 \phi S_2))$; it is assumed that S_1 , $S_2 \in PDL$, $\alpha_1 \in \mathcal{P}(S_1)$ and $\alpha_2 \in \mathcal{P}(S_2)$. The notation cat means "is concatenated into":

$\mathcal{P}(S_1+S_2)) = \{\alpha_1,\alpha_2 | \text{head}(\alpha_1)\underline{\text{cat}}\ \text{tail}(\alpha_2)\}$

$\mathcal{P}(S_1-S_2)) = \{\alpha_1,\alpha_2 | \text{head}(\alpha_1)\underline{\text{cat}}\ \text{head}(\alpha_2)\}$

$\mathcal{P}(S_1*S_2)) = \{\alpha_1,\alpha_2 | \text{tail}(\alpha_1)\underline{\text{cat}}\ \text{tail}(\alpha_2)\}$

$\mathcal{P}(S_1*S_2)) = \{\alpha_1,\alpha_2 | (\text{tail}(\alpha_1)\underline{\text{cat}}\ \text{tail}(\alpha_2)) \wedge (\text{head}(\alpha_1)\underline{\text{cat}}\ \text{head}(\alpha_2$

The graphs of the resulting pictures are illustrated in Figure 3; t and h indicate the tail and head of each expression. Figure 4 uses these operators to describe a line drawing of an "A" and an electrical circuit. Typical members of each primitive class are shown with arrows pointing from the tail to head positions. The lines in the circuit primitives may be any concatenated set of line segments.

The connectivity graph of a PDL expression S and each picture in $\mathcal{P}(S)$ has a tail and head position. tail(S) and head(S) will generaically refer to the tail and head of either a picture described by S or the graph.

Note that $\mathcal{P}(S)$ could be empty. This is the case when the concatenations described by S are not possible according to the definitions of the primitives. For example, if ℓ_1 is a line segment primitive and $\text{head}(\ell_1) = \{(x,y)|x = c_1\}$ and ℓ_2 is another line segment primitive with $\text{tail}(\ell_2) = \{(x,y)|x = c_2\}$, where $c_1 \neq c_2$, then $\mathcal{P}(\ell_1 + \ell_2))$ is empty. However, the graph is constructed by treating ℓ_1 and ℓ_2 abstractly.

The unary operators $\sim$ and / do not describe concatenations but allow the tail and head to be moved. $\sim$ is a tail/head reverser such that tail($(\sim S)$) = head(S) and head($(\sim S)$) = tail(S) . The blanking or superposition operator / works in conjunction with label designators to allow multiple appearances of the same primitive in a description, effectively relocating the tail or head of an expression. The label serves to identify the primitive or structure while the / operator indicates retracing over its associated operand. Figure 5 illustrates these features in describing a complete four-node graph with directed edges.

It is useful to have a notion of description equivalence. The PDL expressions S_1 and S_2 are said to be equivalent (expressed $S_1 \equiv S_2$) if (a) there exists an isomorphism between the graphs of S_1 and S_2 such that corresponding edges have identical names and (b) tail(S_1) = tail(S_2) and head(S_1) = head(S_2) . Label designators on non-primitive PDL expressions are then interpreted as follows:

$$(S_1 \phi_b S_2)^i \equiv (S_1^i \phi S_2^i) \quad , \quad \phi_b \in \{+, -, \times, *\}$$

$$(\phi_u S)^i \equiv (\phi_u S^i) \quad , \quad \phi_u \in \{\sim, /\} \ .$$

Concatenated label designators are treated as single labels. Thus $((a^i + (\sim b))^j + a^i) \equiv ((a^{ij} + (\sim b^j)) + a^i)$.

Any connected picture, in the sense given in the last section, can be described in PDL; in addition, the tail or head can be moved to any node in the graph (or to any primitive tail and head in the picture). PDL also has a number of useful formal properties that allow expressions to be manipulated into convenient forms [19,20].

Primitives are grouped into higher level structures by specifying a grammar $\mathcal{G}$ generating sentences in PDL describing the picture class of interest. $\mathcal{G}$ will be a type 2 (context-free) phrase structure grammar [10] with the restriction that each rule or production is of the form:

$$S \rightarrow pdl_1 | pdl_2 | \ldots | pdl_n \quad , \quad n \geq 1 \; ,$$

where S is a non-terminal symbol and pdl_i is any PDL expression with the addition that non-terminal symbols are allowable replacements for primitive class names. Each grammar $\mathcal{G}$ will have one distinguished non-terminal symbol from which the language $\mathcal{L}(\mathcal{G})$ may be generated; the symbol on the left part of the first production of $\mathcal{G}$ will be the distinguished symbol. Any sentence $S \in \mathcal{L}(\mathcal{G})$ is assumed to have one parse; that is, $\mathcal{G}$ will be an unambiguous grammar.

Let $\mathcal{P}_{\mathcal{G}} = \bigcup_{S \in \mathcal{L}(\mathcal{G})} \mathcal{P}(S)$, the set of all pictures described by the grammar. Then, the hierarchic description $H(a)$ of a picture $a \in \mathcal{P}_{\mathcal{G}}$ with $T_s(a) \in \mathcal{L}(\mathcal{G})$, is defined by the pair

$(H_s(a),(H_v(a))$, where $H_s(a)$ is the parse of $T_s(a)$ according to $\mathcal{A}$ and $H_v(a)$ is a list containing the name, tail, and head of each non-terminal symbol (node of the parsing tree) of $H_s(a)$; the tail and head of a non-terminal symbol is the tail and head of the PDL expression generated by it. A more general system would allow $H_v(a)$ to be the result of obeying semantic rules in 1-1 correspondence with those of $\mathcal{A}$; this "imposed" semantics has not been developed here.

Figure 6 contains a simple example. A more interesting example of a grammar for series/parallel resistance networks is given in Figure 7. Several detailed examples of grammars for pictures in high energy particle physics, characters, text, 2-dimensional geometric objects, and flow charts are given in [19,20].

The complete description scheme can now be given:

1. The class of pictures of interest is described by a grammar $\mathcal{A}$.

2. The description $D(a)$ of any picture $a \in \mathcal{P}_{\mathcal{A}}$ is

$$D(a) = ((T_s(a),T_v(a)),(H_s(a),H_v(a)))\ ,$$

where

$T_s(a) \in \mathcal{L}(\mathcal{A})$,

$T_v(a)$ is a list of the descriptions of all primitives of a ,

$H_s(a)$ is the parse of $T_s(a)$ according to $\mathcal{A}$, and

$H_v(a)$ is a list of the descriptions of all non-terminals in $H_s(a)$.

PICTURE PARSING

The Analysis Problem

The basic information required for the analysis of a set of pictures will be:

(1) A grammar $\mathcal{G}$ defining the pictures $\mathcal{P}_{\mathcal{G}} = \bigcup_{S \in \mathcal{L}(\mathcal{G})} \mathcal{P}(S)$, $\mathcal{L}(\mathcal{G}) \subseteq$ PDL , and

(2) a recognition function for each primitive class named in $\mathcal{G}$.

Then, given a set of pictures $(\alpha_i | i = 1,\ldots,n)$, the pattern recognition task is to discover whether each $\alpha_i \in \mathcal{P}_{\mathcal{G}}$; a more common task, which is often called pattern detection, is to discover whether there exists a $\beta_i \subset \alpha_i$ such that $\beta_i \in \mathcal{P}_{\mathcal{G}}$ — that is, whether some subpicture of α_i is in $\mathcal{P}_{\mathcal{G}}$. The main purpose and important side effect, of a successful recognition is to exhibit the picture description $D(\alpha_i)$ (or $D(\beta_i)$). Our entire analysis process is directed by the description of the picture class — the grammar — and will be called picture parsing.

The primitive recognition mechanism depends on the method of representing pictures and the amount of pre-analysis that is done before parsing. Several possibilities exist:

1. Digitized Pictures

If the pictures are presented for parsing in "raw" or preprocessed digitized form, the recognition functions are picture pattern recognition routines.

2. Representation by a List of Primitives

A list of the names and values of the primitives in a picture might first be obtained by some means external to the parsing system. Then, primitive recognition during parsing occurs by searching these lists.

3. Graph Representation

In a similar manner as number 2 above, a picture might first be represented as a graph with properties associated with the edges. At the primitive level of the parse, graph matching routines could be used to find primitives. This formulation is almost equivalent to several graph isomorphism problems studied by Sussenguth [21] — Is a graph G isomorphic to another graph G' , to a subgraph of G' , to a partial graph of G' or to a partial subgraph of G' ? In the picture case, G is the graph of some member of $\mathcal{L}(\mathcal{U})$ and G' is the graph of the picture under consideration.

4. PDL Primitive Descriptions

The input to the parse is a PDL primitive description, perhaps obtained manually or, as a more interesting case, as the output of a generation procedure. Since , in general, many PDL descriptions are possible for the same picture, a string analysis based on $\mathcal{A}$ would often fail, even if the picture were in $\mathcal{Q}_{\mathcal{A}}$. However, a PDL expression can be easily transformed into a graph or a list of primitives and the recognition treated as in number 2 or 3 above.

The last three examples are variations of each other and could be handled by the same primitive recognition system, either graph matching or list searching.

Most of the analysis superstructure will be applicable to a variety of picture representations. The most interesting, challenging, and practical parsing deals with the digitized picture directly; this has been our main emphasis. One of the features of this approach to analysis is that the primitives in a digitized picture can often be recognized more easily than in ad hoc methods.

Goal-Oriented Picture Parsing

String language analyzers that employ the syntax explicitly are usually called syntax-directed [7]. Many of the same techniques are used for picture parsing. We first review some basic concepts in one-dimensional string analysis and then extend these to handle two- and three-dimensional pictures.

1. Syntax Analysis of String Languages

Assume that P is the distinguished symbol of a grammar $\mathcal{G}$. A bottom-up parse of a string s starts with s and attempts to reduce it to P by reverse applications of the productions of $\mathcal{G}$. A top-down parse does the opposite; starting with P, one searches for a series of productions that eventually generate s. In the first case, the parsing tree is built from the leaves to the root P; in the latter, the tree is formed successively from the root. The same tree

is built in either case if $s \in \mathcal{L}(\mathcal{A})$ and $\mathcal{A}$ is unambiguous; Figure 8 illustrates the tree formation for both types. Most syntax-directed compilers use a combination of these.

A bottom-up scheme will read several symbols and try to reduce them as far as possible before continuing; when no more reductions can be made to a substring, the next terminal symbol is read, composed, and returned to an input routine, say GETNEXTSYMBOL(loc) , where loc is a pointer to the location of the next symbol in the input string. The top-down method is goal-oriented or predictive in nature. For example, an analyzer for the grammar of Figure 8(a) would initially call a routine to find a P ; the P routine would look for the input string "(start +" and then call the routine TRACK if successful; TRACK would first look for "beam" as the next set of input symbols; if "beam" was not present, TRACK would then look for "(neg +" and call the routine PRS if successful. This process continues until either the P routine is successful after reading the entire input string, or P fails. At each stage, each element of the right part of a production becomes a goal or prediction for the analyzer. When a goal is a terminal symbol, the parser usually calls a logical or Boolean routine, say LOOKFORSYMBOL(name,loc) ; the routine returns true if the next input symbol (at loc) is equal to name, and false otherwise. Alternatives in productions often cause false goals to be generated and the analyzer must back-up and try

again with another alternative. Both systems parse from left to right.

2. Syntax Analysis of Pictures

There exist picture processing analogs of these basic language parsers. The "terminal" symbols of the input picture α are just the picture primitives contained in α. The entire purpose of the parse is to recognize these primitives — a pattern recognition task comparable to the recognition of terminal symbols by the input devices of computers — and group them into the structures described in $\mathcal{A}$.

Given a grammar $\mathcal{A}$ with distinguished symbol P, a bottom-up analysis of α would probably start with a small connected set of primitives of α and attempt to combine them according to $\mathcal{A}$. (There is the problem of where to look for the first few primitives.) When a new primitive is required, a routine GETNEXTPRIMITIVE(loc) is called, where loc is a list of two- or three-dimensional picture pointers; loc would depend on the tails and heads of the previous primitives found. The routine would return the description of a primitive found at some location of loc or an indication that no primitive was located there. Unfortunately, the entire pattern recognition mechanism would have to be incorporated at each of these calls since any primitive could be at loc; also, any number of primitives could appear at these locations and the "wrong" one might be found. Searching for blank primitives in such a system would be extremely difficult. The grammar

could be used to produce a reduced list of possible primitives at each point; however, this would result in a goal-directed system that is much more complex than the pure one discussed next. For these reasons, this approach was not taken. (The above arguments are not quite as significant if the primitives were recognized before-hand and the picture was represented as a list of primitives, a graph, or a PDL expression; however, this would be pushing the most difficult problem outside of the parsing system.)

Our pure top-down or goal-oriented analyzer starts with P and attempts to generate from left to right, a sentence $S \in \mathcal{L}(\mathcal{G})$ such that $S = T_{\mathcal{G}}(\alpha)$. When the goal is a primitive class name, the routine LOOKFORPRIMITIVE(name,loc) is called, where loc specifies the coordinates of the tail, head, or both of name, depending on the concatenations expressed in the production containing or leading to name; LOOKFORPRIMITIVE in turn calls one pattern recognition routine whose sole purpose is to determine whether a member of $\mathcal{P}$(name) is located at loc in the picture. If the recognition is successful, the value of the primitive is returned.

Explicit top-down analyzers for the syntax of Figure 8(a) illustrate the approach; the method for string analysis is essentially that of Leavenworth [12]. The propositional connectives $\wedge$("and") and $\vee$ ("or") are to be interpreted from left-to-right in the McCarthy sense [13]; i.e. $A \wedge B$ means

if A **then** B **else false** and A ∨ B means **if** A **then true** **else** B .

Top-Down String Parser for $\mathcal{A}$ of Figure 8

```
Boolean Procedure P;
   P := Lfs('(') ∧ Lfs('start') ∧ Lfs('+') ∧ TRACK ∧ Lfs(')');
Boolean Procedure TRACK;
   TRACK := Lfs('beam')∨(LFS('(')∧Lfs('neg')∧Lfs('+')∧PRS∧Lfs(')'))
Boolean Procedure PRS;
   PRS := (Lfs('(') ∧ PAIR ∧ Lfs('+') ∧ PRS ∧ Lfs(')')) ∨ PAIR;
Boolean Procedure PAIR;
   PAIR := Lfs('(')∧Lfs('pos')∧Lfs('#')∧Lfs('neg')∧Lfs(')');
```

Lfs(name) , the LOOKFORSYMBOL routine, returns **true** if the next symbol in the output string is name , and **false** otherwise. For simplicity, we have omitted the string pointer and its administration.

If we interpret the grammar of Figure 8(a) as describing a set of pictures $\mathcal{P}_{\mathcal{A}}$ (Figure 8(b)), our top-down picture analyzer works as follows:

```
Boolean Procedure P(t,h);
   P := Lfp('start',t,hs) ∧ TRACK(hs,h);
Boolean Procedure TRACK(t,h);
   TRACK := Lfp('beam',t,h) ∨ (Lfp('neg',t,hm) ∧ PRS(hm,h));
Boolean Procedure PRS(t,h);
   PRS := (PAIR(t,hp) ∧ PRS(hp,h)) ∨ PAIR(t,h);
Boolean Procedure PAIR(t,h);
   PAIR := Lfp('pos',t,h1) ∧ Lfp('neg',t,h);
```

Lfp(name,t,h) , the LOOKFORPRIMITIVE procedure, calls a

pattern recognition routine to look for a member of $\mathcal{P}$(name) with tail located at t . If successful, it returns true and sets h to the head coordinates of the found primitive.

The parameter t for the routines P , TRACK , PRS , and PAIR is an input tail location that the goal must satisfy; if the routine is successful, the parameter h will be set to the head of the goal by these procedures. t and h are sets of coordinates defining points or neighborhoods in n-space. The parser is executed by the call: P(org,Ω) , where org is the picture origin and Ω denotes undefined.

A pure goal-oriented parser was selected for the PDL analysis system for the following reasons:

1. The language portion of the analysis (stepping through the grammar) is conceptually very simple.
2. The syntax directly expresses the algorithm for analysis.
3. It was conjectured (and verified later in our experimental work) that any inefficiencies due to the back-up caused by false goals would be insignificant compared to the primitive recognition time. Once a primitive is recognized, it is stored; thus, if a goal fails, its primitives may be used later in the analysis without rerecognition.
4. Goal-oriented analysis is beneficial for primitive recognition. Each primitive recognizer could include its own preprocessing and often need not be as precise as a scheme that requires a search for all primitives at any point in the analysis. The same advantages hold over global methods that produce a list of all the

primitives in a picture. These advantages are achieved because the concatenation operators in conjunction with the previously found primitives tell the system where to look for the next primitive.

As well as the use of two- or three-dimensional tail and head pointers, there are a number of other interesting differences between string and picture parsing that must be taken into account in a general parser:

1. Multiple recognition

Consider a picture search for primitives that satisfy the expression: $(a + (b \times b))$. The first element of $\mathcal{P}(b)$ may be recognised twice unless it is eliminated from the picture after it has been found; the elimination procedure may be very complex when patterns overlap.

2. Commutative Expressions

The topological commutativity of the $\times$ and $*$ operators can result in recognition problems when the initial expressions of the operands are identical. The expression $(a + ((b + c) \times b))$ can be represented by the tree of Figure 9(a). If the first b found is the one of the left branch of the tree, then the parse would try to find a c adjoined to it and fail. One solution is to change the strictly left-to-right recognition if a failure occurs in a commutative expression; in the above example, the search could then back-up after the failure and try $(b \times (b + c))$, remembering that the head of the expression is to be at the first b. A simpler

solution, when this confusion is possible, is to write the syntax so that both expressions appear. For example:

$$A \rightarrow (B * C) | (C * B)$$

$$D \rightarrow (E \times F) | ((F \times E)^{2} + (/((\sim E) + F)^{2}))$$

3. Detection

A similar difficulty can occur in a detection problem. If the parse is looking for $(a + (a + (a \times a)))$ in a picture whose primitives have the tree of Figure 9(b), then a search that follows the right branch of the tree will fail. Here an elaborate back-tracking procedure would be necessary.

The difficulties of number 2 and 3 are similar to those that occur in the graph matching (isomorphism) problems mentioned earlier. In general, there is no "optimum" technique to handle them; the worst case requires an exhaustive search of all possible paths in the graph.

4. Recursion

In string parsing, there is an identifiable beginning and end of the input string; the picture analogs are the origin and an empty picture. For real pictures, the latter is not identifiable since even after a successful analysis, there will be much noise and extraneous data in a picture. Consider the syntax:

$$S \rightarrow a | (a + S)$$

Parsing success would occur on the recognition of the first element of $\mathcal{A}(a)$ regardless of whether more a's were concatenated onto it. This problem disappears for simple right recursions if they appear as the first alternative of a production.

5. The Effect of - and $\sim$

Since the expression (A - B) can be rewritten equivalently as (A + (($\sim$B) × λ)) , only $\sim$ is considered. An appearance of $\sim$ interrupts the left-to-right flow through the right parts of a production. For example, a search for a picture part satisfying (a + ($\sim$(b + c))) would require finding the primitive c after a is recognized; the parser could make the transformation (a + ($\sim$(b + c))) ≡ (a + (($\sim$c) + ($\sim$b))) and use the latter. This could be done more easily by transforming the grammar before starting the parse so that $\sim$ only applied to primitives.

These problems have to be treated in a completely general parser. For the work reported here, some are ignored and others handled by simple changes to the syntax; this will be noted when discussing the general parser and the implemented system.

Explicit language and picture analyzers of the type illustrated earlier require writing a new set of procedures for each grammar. General language parsers that accept grammars as input and automatically produce the equivalent of the

explicit parsing procedures have been written and used successfully [7]. The same type of system has been developed by this writer for picture parsers. The next section discusses the general picture parsing algorithm on which our implemented system is based.

A GENERAL PDL PICTURE PARSING ALGORITHM

The parsing is considerably simplified if the grammar $\mathcal{G}$ is first transformed into a standard form, $\mathcal{G}_{sf}$, called the PDL standard form of the grammar. Each rule of $\mathcal{G}_{sf}$ will have one of the following forms:

$A \rightarrow (B \phi C)$
$A \rightarrow D$
$A \rightarrow (/p)$
$A \rightarrow (\sim p)$
$A \rightarrow (\sim(/p))$

where $\phi \in \{+, \times, *\}$, A is a non-terminal symbol, B and C are non-terminal symbols possibly with labels, D is a non-terminal symbol or primitive class name (either one possibly labeled), and p is a primitive class name with or without label. In general, the following relations exist between $\mathcal{G}$ and $\mathcal{G}_{sf}$:

1. $Q_{\mathcal{G}} = Q_{\mathcal{G}_{sf}}$
2. $\mathcal{L}(\mathcal{G}) \neq \mathcal{L}(\mathcal{G}_{sf})$
3. There is a 1-1 correspondence between sentences of $\mathcal{G}$ and $\mathcal{G}_{sf}$ such that $x \equiv y$ if $x \in \mathcal{L}(\mathcal{G})$ corresponds to $y \in \mathcal{L}(\mathcal{G}_{sf})$.

Examples:

1. $\mathcal{G}$: $A \rightarrow (\sim B) \mid B$
   $B \rightarrow (c \cdot d)$

   $\mathcal{G}_{sf}$: $A \rightarrow B_{\sim} \mid B$
   $B_{\sim} \rightarrow (T_{\sim d} \cdot T_{\sim c})$
   $B \rightarrow (T_d \cdot T_c)$
   $T_{\sim d} \rightarrow (\sim d)$
   $T_{\sim c} \rightarrow (\sim c)$
   $T_d \rightarrow d$
   $T_c \rightarrow c$

2. $\mathcal{G}$: $A \rightarrow ((b \cdot c) - d)$

   $\mathcal{G}_{sf}$: $A \rightarrow (S_{b \cdot c} \cdot S_{(\sim d) \times \lambda})$
   $S_{b \cdot c} \rightarrow (T_b \cdot T_c)$
   $T_b \rightarrow b$
   $T_c \rightarrow c$
   $S_{(\sim d) \times \lambda} \rightarrow T_{\sim d} \times T_\lambda$
   $T_{\sim d} \rightarrow (\sim d)$
   $T_\lambda \rightarrow \lambda$

3. $\mathcal{G}$: $A \rightarrow (b \cdot (\sim A)) \mid b$

   $\mathcal{G}_{sf}$: $A \rightarrow (T_b \cdot A_{\sim}) \mid b$
   $A_{\sim} \rightarrow (A \cdot T_{\sim b}) \mid (\sim b)$
   $T_{\sim b} \rightarrow (\sim b)$
   $T_b \rightarrow b$

An algorithm for transforming an arbitrary PDL grammar into standard form is presented in the appendix. We also state,

without proof, that a syntax analysis on the transformed grammar can be converted to one for the original. Our parsing algorithm can then assume, with no loss of generality, that the input grammar is in standard form.

The difficulties mentioned in the last section are handled in a straightforward way. The primitive recognizers are assumed to eliminate a picture on a successful recognition so that multiple recognition is not possible. The problems of commutative expressions and recursion are resolved by suitable changes to the syntax as suggested. No provision is made for the detection difficulty. The reversal of the left-to-right scan caused by the - and ~ operators can no longer occur when the grammar is in standard form. Finally, left-recursive productions are not allowed. This is to be interpreted as left recursion in the picture sense. For example,

$$A \rightarrow (((A + b) + c) + d) | e$$

is left recursive in the picture sense since the parentheses are only used for grouping; this is also the case for:

$$A \rightarrow ((\lambda + (\lambda + \lambda)) + (A + b)) | c$$

The last restriction is only made because left-recursive productions have to be treated as a special case in top-down analyzers to prevent the general algorithms from getting into infinite loops.

A push-down stack is used to store the picture syntax tree and goals for the parser. If $S \rightarrow s_1|s_2|\ldots|s_j|\ldots|s_m$ is a rule in $\mathcal{A}$ and S becomes a goal during the analysis, we will call the right part s_j the jth alternative of S. If s_j is of the form $(s_{j_1} \phi s_{j_2})$, s_{j_1} and s_{j_2} will be called the left and right successors of S; otherwise s_j is the successor of S. Each stack element will have 7 components defined as follows:

Goal[s]: A non-terminal symbol of $\mathcal{A}$ or a primitive expression of the form p, (/p), (~p), or (~(/p)), where p is a primitive class name. (Actually, a pointer to these symbols.) A label may be associated with this element.

Alt[s]: An alternative number for a rightpart of Goal[s].

Lsuc[s]: A stack pointer to either the left successor or the successor goal of Goal[s].

Rsuc[s]: A stack pointer to the right successor of Goal[s].

Con[s]: The tail or head constraints for Goal[s]. If Cons[s] = TH, then both the tail and head are constrained to tail[s] and head[s]. Otherwise, Cons[s] = T indicating only the tail is constrained.

tail[s]: The tail coordinates of Goal[s].

head[s]: The head coordinates of Goal[s].

The procedure Stack is defined:

```
integer procedure Stack(G,A,L,R,C,t,h);
      value G,A,L,R,t,h; integer G,A,L,R,t,h;
begin
   Stack := s := s+1;
   Goal[s] := G; Alt[s] := A;
   Leuc[s] := L; Reuc[s] := R; Cone[s] := C;
   tail[s] := t; head[s] := h
end Stack
```

s is the current top of the push-down stack and is global to the parsing procedures.

The following Algol-like algorithm will perform a goal-oriented parse of a picture given the above form of grammar. We use a main procedure "Parse" calling two other procedures "Nextalt" and "Tryagain".

```
Boolean procedure Parse(g); value g; integer g;
comment g is a stack pointer to the current goal, Parse returns
        true if the goal is satisfied and false otherwise. On
        a successful parse, the stack will contain the picture
        syntax tree;
begin
   integer R,t1,hd,L,r; Boolean b, leftb; string op;
   R := Rightpart(Goal[g], Alt[g]);
   if Primitive(R) then
   begin
      comment R is a primitive;
      if Lookforprimitive(R,g,t1,hd) then
      begin
         comment R has been successfully recognized;
         Parse := true; head[g] := hd;
         Leuc[g] := Stack(R,0,0,0,Con[g],t1,hd)
      end
      else Parse := Nextalt(g)
   end
   else
   if Nonterminal(R) then
   begin
      comment R is a non-terminal symbol;
      L := Stack(R,1,0,0,Con[g],tail[g],head[g]);
      if Parse(L) then
      begin
         comment R has been successfully recognized;
         Parse := true; Leuc[g] := L; head[g] := head[L]
      end
      else Parse := Nextalt(g)
   end
   else
```

```
begin
   comment R is of the form  (S1 ØS2) or (S1 ØS2)*;
   op := Catop(R); b := false; leftb := true;
   ℓ := Stack(Leftop(R),1,Ω,Ω,
          if(op = '*') ∧ (Con[g] = TH) then TH else F,
          tail[g], if op = '*' then head[g] else Ω);
   leftb := Parse(ℓ);
   while leftb ∧ ¬b do
   begin
      comment S1 has been recognized;
      r := Stack(Rightop(R),1,Ω,Ω,
             if(Con[g] = TH) ∨ (op = '*') then TH else F,
             if op = '+' then head[ℓ] else tail[ℓ],
             if op = '*' then head[ℓ] else head[g]);
      if ¬Parse(r) then
      begin
         comment S2 has failed, Tryagain on S1;
         s := r-1;
         leftb := Tryagain(ℓ)
      end
      else b := true
   end
   if b then
   begin
      Leuc[g] := ℓ; Reuc[g] := r; head[g] := head[r];
      Parse := true
   end
   else Parse := Nextalt(g)
end
end Parse
```

```
Boolean procedure Nextalt(g); value g; integer g;
comment g is a stack pointer to the current goal.  Nextalt
        returns true if the next alternative rightpart of g
        is successfully parsed, and false otherwise;
begin
   s := g;
   if Alt[g] = Numberofalternatives(g) then Nextalt := false
   else
   begin
      Lsuc[g] := Rsuc[g] := Ω;
      Alt[g] := Alt[g]+1;
      Nextalt := Parse (g)
   end
end Nextalt

Boolean procedure Tryagain(g); value g; integer g;
comment g is a stack pointer representing the root of a picture
        syntax tree.  Tryagain systematically attempts to get
        another successful parse by trying the next alternative
        rightpart of each non-terminal symbol (interior node)
        of the tree.  Nodes are visited in the reverse order
        in which they were generated.  Tryagain returns true
        if another parse is obtained; otherwise, it returns
        false;
begin
   integer Gr,Cr,t; string op; Boolean b,leftb;
   if Rsuc[g] = Ω then
   begin
      comment Lsuc is either a primitive or a non-terminal symbol
      if Primitive(Lsuc[g]) then
            Tryagain := Nextalt(g)
      else
      if Tryagain(Lsuc[g]) then
```

```
   begin
      Tryagain := true; head[g] := head[Leuc[g]]
   end
   else Tryagain := Nextalt(g)
end
else
if Tryagain(Reuc[g]) then
begin
   comment A new parse has been found for the right successor
           of g;
   Tryagain := true; head[g] := head[Reuc[g]]
end
else
begin
   comment Try another parse for g's left successor;
   Cr := Goal[Reuc[g]]; Cr := Con[Reuc[g]];
   s := Reuc[g]-1; b := false; leftb := true;
   op := Catop(Rightpart(Goal[g],Alt[g]));
   while leftb ∧ ¬b do
   begin
      if Tryagain(Leuc[g]) then
      begin
         comment Parse g's right successor;
         Reuc[g] := t := Stack(Cr,1,Ω,Ω,Cr,
            if op = '+' then head[Leuc[g]] else tail[Leuc[g]],
            if op = '*' then head[Leuc[g]] else head[g]);
         b := Parse(t)
      end
      else leftb := false
   end
   if b then
   begin
      Tryagain := true;
      head[g] := head[t]
   end
   else Tryagain := Nextalt(g)
end
```

The auxiliary procedures called by Parse perform the following functions:

1. Rightpart(x,n):
   Rightpart returns the nth right part of the production whose left part is the non-terminal symbol S , where x = S or S^{ℓ} . If x is labeled, the same label is adjoined to the right part.

2. Primitive(x):
   x = y or y^{ℓ} , Primitive returns <u>true</u> if y is of the form: p or (/p) or (∿(/p)) , where p is a labeled or unlabeled primitive class name.

3. Lookforprimitive(x,g,t,h):
   x is of the same form as the parameter of Primitive. g is a stack location of the superior goal of x . Lookforprimitive calls a pattern recognition routine that attempts to find the primitive of x at the location defined by tail[g] and head[g]. If Con[g] = T , tail[g] is the only constraint on the primitive; if Con[g] = TH , both tail and head are constrained. A successful search will return <u>true</u>, and the tail location t and head location h of x . If the primitive of x is labeled, Lookforprimitive first checks to see if it was found previously; if so, then a picture recognition is not necessary. A ∿ in x indicates that the head of the primitive must be at tail[g] and (possibly) the tail at head[g]. Lookforprimitive returns <u>false</u> if an x is not found satisfying the given constraints.

4. Nonterminal(x):
   This procedure is <u>true</u> if x = S or S^{ℓ} where S is a non-terminal symbol; otherwise Nonterminal returns <u>false</u>.

5. Catop(x), Leftop(x), Rightop(x);

   x is of the form $(S_1 \phi S_2)$ or $(S_1 \phi S_2)^{\ell}$, where S_1 and S_2 are non-terminal symbols possibly with labels and $\phi \in \{+, \times, *\}$. Then Catop(x) = ϕ, Leftop(x) = S_1 or S_1^{ℓ} and Rightop(x) = S_2 or S_2^{ℓ}.

6. Numberofalternatives(x);

   x = S or S^{ℓ} where S is a non-terminal symbol. Numberofalternatives returns the number of rightparts of the production whose left part is S.

To analyze a picture, the following sequence of statements is executed:

```
s := 0;
Stack(D,1,Ω,Ω,T,origin,Ω);
b := Parse(s);
```

where D is the designated symbol of the grammar and origin is the given origin coordinates (tail). On a successful analysis, head[1] will contain the picture head coordinates (head(D)). The parsing tree can be easily presented by tracing through the stack.

The algorithm is remarkably similar to a restricted form of a classical top-down language analyzer; however, there are several major differences:

1. Instead of a one-dimensional string pointer, we use a pair of pointers, a tail pointer and a head pointer.

2. The algorithm is totally top-down; even the terminals (the primitives) are recognized in a goal-directed manner. This is especially significant in picture processing. In a conventional system, we are always asking the question "What pattern is there?" and are therefore forced to apply the entire pattern recognition system at each step, whereas above, we are directed by the grammar to ask "Is a pattern of class x there?", where both the general location and name of x derives from the grammar and the prior results of the analysis. a priori knowledge of the picture class is reflected in the grammar and used by the algorithm.

3. The algorithm is independent of the dimension of the picture. The dimension $n (n \geq 1)$ is given by the number of coordinates in the tail and head positions. In fact, if we restrict our operators to + , remove parentheses, and do not allow labels, we have a top-down context-free grammar analyzer, where the grammar is in Chomsky normal form [10].

SOME IMPLEMENTATION RESULTS

The Implemented System

The primary purpose of the implementation was to evaluate this approach to picture processing by actually analyzing a non-trivial set of real pictures. A class of spark chamber pictures produced in high energy particle physics experiments was selected. These pictures can be described in PDL using the operators + , × , and * , and without labels.

The parser has been programmed for a subset of the PDL language called SPDL (Simple Picture Description Language).

SPDL is restricted to the operator set $(+,\times,*)$ with no labels, but is otherwise identical to PDL. Most of the discussion preceding the algorithm of the last section applies to the SPDL parsing system also.

A schematic of the system organization is shown in Figure 10(a). The SPDL syntax analyzer or parser is a general purpose program. For a particular application, a set of primitive recognition routines is added (or used from a library) and the defining picture grammar is input as data. The area enclosed by dotted lines in the figure is the complete analysis system. Two- or three-dimensional pictures can be handled without any changes to the programs. The system was programmed entirely in FORTRAN IV (employing some library assembly language routines for the display) and run on an IBM 360, Model 50, with 2250 Display Unit under the OS/360 operating system.

Figure 10(b) contains a general flow chart of the program. The input productions of $\mathcal{G}$ are of the form: $A \rightarrow (B \phi C)$ or $A \rightarrow D$, where A is a non-terminal symbol, D is a non-terminal symbol or primitive class name, $\phi \in (+,\times,*)$, and B and C may be <u>any</u> SPDL expression composed of primitive class names and non-terminal symbols; left-recursive rules are not allowed. Each production is first parsed (in the language sense) for well-formedness and then converted to standard form. The origin is a coordinate triple (x,y,z) defining the start point for all SPDL descriptions; z is marked as undefined for 2-dimensional pictures.

The main loop successively reads and parses pictures. The picture input data consists of the coordinates of those parts of the picture whose light intensity is less than a given threshhold; film is digitized by using a flying spot scanner [15]. The core of the system is the parser which is an iterative (non-recursive) version of the algorithm presented in the last section. On a successful parse of a picture α, the program prints a stack representation of the parsing tree and semantics (the hierarchic description $(H_s(\alpha), H_v(\alpha))$), and the primitive value list $T_v(\alpha)$. We also maintain a "failure" list of primitives found but later discarded because of false goals; thus, a particular primitive need be recognized only once. The failure list is always searched before calling the digital pattern recognition routines. The 2250 cathode-ray tube display visually shows the evolution of the parse; the residue picture (the original minus the eliminated primitives) and an abstract version of the recognized picture are continuously displayed with markers pointing to the tail and head of the last primitive found. This proved to be extremely useful for evaluating the system and for debugging.

Primitive Recognition

The primitive pattern recognition routines are called by the parser with coordinates defining the required location of the tail or head of the primitive (see the earlier discussion explaining the Lookforprimitive procedure). To

recognize patterns in real digitized pictures, we include in each recognition routine a definition of a neighborhood of points $N(x)$ for a tail or head point x. Then any point $y \in N(x)$ is considered equivalent to x for satisfying the concatenation requirements. For example, suppose $N_q(x)$ defines a neighborhood for the tail point of a primitive q. then, for practical purposes, we define

$$\mathcal{P}(S+q)) = \{\alpha \mid \alpha = \alpha_1 \cup \alpha_2, \alpha_1 \in \mathcal{P}(S) \wedge \alpha_2 \in \mathcal{P}(q) \wedge tail(\alpha_2) \in N_q(head(\alpha_1))\}.$$

The definition of a blank or don't care primitive must usually include a characterization of the set of primitives than may be concatenated onto its tail and/or head, as well as the geometric constraints on their relative locations. Assume that one is parsing a picture according to the grammar:

$$P \rightarrow (r + (b + S))$$
$$S \rightarrow s1 \mid s2 \mid \ldots \mid sn$$

where r, $s1, s2, \ldots, sn$ are visible primitives and b is a blank primitive. Then the b recognizer must discover the presence of some member of $\mathcal{P}(S) = \bigcup_{i=1}^{n} \mathcal{P}(si)$; often, this search can be reduced to finding a feature that is common to all members of $\mathcal{P}(S)$ and is unique to $\mathcal{P}(S)$ in the picture. The same statement can frequently be made when $n = 1$ in the above grammar; in the worst case, the recognition of b would involve recognizing and obtaining a complete description of

s1 . If the latter occurred, the recognition function for s1 would be vacuuous and always return true.

Example 1:

Consider the primitive start in the syntax of Figure 8. Its purpose is to find the tail of some member of $\mathcal{P}$(beam) or $\mathcal{P}$(neg) entering at the left edge of the picture. This can be done easily by using a simple line recognizer locally in regions near the left edge to find the beginning of a track; once this is accomplished, the recognizers for either beam or neg can determine its extent, curvature, and perhaps, point density.

Example 2:

Consider the problem of recognizing sentences of printed text in terms of characters and words. Each character within a word is separated from its succeeding character by some inter-character spacing; we will call this intercharacter spacing the blank primitive ics . Similarly, words are separated by a blank interword spacing primitive iws . Assuming a sentence appears on one line, the following grammar will describe a sentence:

SENTENCE → (WORD + (ics + .))|(WORD + (iws + SENTENCE))
WORD → CHAR|(CHAR + (ics + WORD))
CHAR → A|B|C|...|Z

The recognition routines for both ics and iws have only to find the beginning (tail) of a letter or "."; this can be done by counting point densities along a line or by some other simple global test. The recognizers for the characters do the actual classification.

Blank and don't care primitives are used extensively in the application described in the next section. These recognizers are all based on the ideas described above.

Spark Chamber Film Analysis by the SPDL System

The SPDL system was applied to the analysis of spark chamber film produced in a high energy physics experiment, the "colliding beam" experiment, conducted at Stanford University [3]. The purpose of the physics experiment was to measure the angular distribution of electron-electron scattering at an energy level of 600 MeV and over the angular range from 40° to 90°. Electron beams were supplied by the Stanford Mark III linear accelerator. These were circulated in opposite directions in two storage rings having a common section; electrons from the two rings collide in the common section and scatter in opposite directions. The scattering event is observed via a set of spark chambers and counters through which the electrons then pass. Each scattered electron traverses successively through a 6 gap chamber, a 4 gap chamber, and a shower chamber.

A schematic of the film format is illustrated in Figure 11. The side and front views appear on the left and right halves respectively. The interaction points are along the central dotted horizontal. A possible event is indicated by a linear track of sparks through the upper 6 and 4 gap chambers and a similar track through the corresponding lower chambers. It is possible for several events and an arbitrary number of sparks to appear in the chambers. The configuration of sparks in the shower chambers are used to assist in the identification of the particle types. The central portion of the film contains

a data box with digits to the left and coded version of these to the right. Information in the data box includes the frame number (the top leftmost 4 digits), roll number, electron beam phase, and date; the first 4 digits in the coded box is the frame number. The "X"s beneath some of the chambers are fiducial markers which are chamber engravings whose precise positions are known; fiducials allow 3-dimensional reconstruction of the experiment in real space.

Figure 12 contains a typical photograph after it has been digitized and displayed on the 2250. The collinear sparks in the 4 gap and 6 gap chambers indicate the presence of an interesting event. A plot of a small "window" of a typical digitized picture of this class is illustrated in Figure 13; the clustered points are sparks while the isolated ones are background noise. Each photograph produced about 1800 digitized points; more than half of the points were not noise.

This picture class is an excellent test of the SPDL system for the following reasons:

1. The pictures are not contrived. They are real pictures produced in a physics experiment with no a priori thoughts about using the SPDL system for this analysis.
2. There is a large amount of detail in each picture.
3. The pictures are well structured.
4. While the photography was very good, the inaccuracies, errors, and noise that are common to most pictures appear here also. Some of these are due to the variation of intensity of the picture components, the nonuniformity of sparks, slipping of the cameras mirrors

between frames, occasional malfunction of parts of the data box, errors of digitisation, and distortions introduced by the flying spot scanner,

5. The pictures are representative of one class of particle physics pictures that is produced in great quantity and requires detailed analysis; this remark applies to the forseeable future,

The following grammar was used to describe these pictures:

```
CLBM → (STRT + ( CDB × (FRVW × SDVW ) ) )
CDB  →  (BBND × (IDS + (DATA + (IDS + BBND)) ) )
DATA →   ( DIGT×(IDS + DATA )) | (DIGT×NULL)
SDVW →  ( ( XSB + CSB ) × ( XST + CST ) )
FRVW → ( ( XFB + CFB ) × ( XFT + CFT ) )
CST  →  (ST46 × SHST )
CFT  →  (FT46 × SHFT )
CSB  →  (SB46 × SHSB )
CFB  →  (FB46 × SHFB )
SB46 →   NOSB | (LO4×LO6)
FB46 →   NOFB | (LO4×LO6)
LO4  →   ((B4GL+CLO)×LO4D) | NULL
LO4D →   LO4 | NULL
CLO  →   EVNT | SPRK
SPRK →   NULL
LO6  →   ((B6GL+SPRK)×LO6D) | NULL
LO6D →   LO6 | NULL
ST46 →  NOST | (HI6 ×HI4)
FT46 →  NOFT | (HI6 × HI4)
HI4  →  ((B4GH+SPRK)×HI4D) | NULL
HI4D →  HI4 | NULL
CHI  →  EVNH | SPRK
HI6  →  ((B6GH + CHI )×HI6D ) | NULL
HI6D →  HI6 | NULL
```

The picture structure described by each non-terminal symbol is:

CLBM: a colliding beam experiment picture
CDB: coded data box
DATA: contents of the data box

$\left\langle \begin{matrix} SD \\ FR \end{matrix} \right\rangle$ VW: $\left\langle \begin{matrix} side \\ front \end{matrix} \right\rangle$ view

C $\left\langle \begin{matrix} ST \\ FT \\ SB \\ FB \end{matrix} \right\rangle$: contents of $\left\langle \begin{matrix} side \\ front \\ side \\ front \end{matrix} \right\rangle$ view chamber, $\left\langle \begin{matrix} top \\ top \\ bottom \\ bottom \end{matrix} \right\rangle$ half of picture

$\left\langle \begin{matrix} ST \\ FT \\ SB \\ FB \end{matrix} \right\rangle$ 46: $\left\langle \begin{matrix} side \\ front \\ side \\ front \end{matrix} \right\rangle$ view, $\left\langle \begin{matrix} top \\ top \\ bottom \\ bottom \end{matrix} \right\rangle$ half, 4 and 6 gap chambers

$\left\langle \begin{matrix} LO4,LO4D \\ LO6,LO6D \\ HI4,HI4D \\ HI6,HI6D \end{matrix} \right\rangle$: $\left\langle \begin{matrix} low \\ low \\ high \\ high \end{matrix} \right\rangle$ half of picture, $\left\langle \begin{matrix} 4 \\ 6 \\ 4 \\ 6 \end{matrix} \right\rangle$ gap chamber

C $\left\langle \begin{matrix} LO \\ HI \end{matrix} \right\rangle$: either a spark in a $\left\langle \begin{matrix} low \\ high \end{matrix} \right\rangle$ 4 gap chamber or an event

SPRK: a spark

The parsing sequence is defined by the grammar. The data box is first found (since it is an obvious and easily recognized pattern), and the front and side views are then analyzed relative to its tail (CLBM). The data box (CDB) is described as a left boundary (BBND) followed by an arbitrary number of digits (DATA) and a right boundary (BBND); the number of digits is actually fixed at 22, but it is more convenient to use a recursive production. The front and side views are defined in terms of a bottom part ((XFB + CFB),(XSB + CSB)) and a top part ((XFT + CFT),(XST + CST)) where (XFB,XSB,XFT,XST) are

the fiducial X's ; an arrow in Figure 11 points to each of the 4 fiducials used. In each of the four parts, or quadrants, of the picture, the grammar divides the analysis into two — the contents of the 4 and 6 gap chambers (ST46,FT46,SB46,FB46) , and that of the shower chambers (SHST,SHFT,SHSB,SHFB) . The 4 and 6 gap chambers may contain sparks (SPRK) and events (EVNT,EVNB) . Recursive productions for the chamber descriptions ((LO4,LO4D),(LO6,LO6D),(HI4,HI4D),(HI6,HI6D)) indicate that an arbitrary number of sparks and events can be present, LO4D,LO6D,HI4D,HI6D , and, often, the null point primitive NULL are used to avoid excessive backtracking during the analysis.

Figure 14 is a photograph of the 2250 display containing the "graph" of the recognized or successfully parsed picture of Figure 12; superimposed on this is the name associated with each primitive. The primitives have the following meanings.

BBND	:	data box boundary
DIGT	:	digit of data box
IDS	:	interdigit distance
ORIGIN	:	origin of digitized coordinate system
STRT	:	a blank primitive pointing to the data box
{XFB,XSB,XST,XFT}	:	fiducial X's , one in each quadrant: F = front view , S = side view T = top view , B = bottom view
EVNT	:	an event in the lower part
EVNB	:	an event in the upper part
{B4GL,B6GH}	:	blank primitives pointing to the first spark in lower 4 gap and upper 6 gap chambers respectively
{SHSB,SHFB,SHST,SHFT}	:	the spark contents of each shower chamber

Ten photographs were digitized and then "successfully" analyzed — the criterion of success being whether human viewers agreed with the classifications. Although the experimental sample is relatively small (typically, thousands and hundreds of thousands of photographs are produced in one high energy physics experiment), several significant points were illustrated by our analysis:

1. After the general analysis algorithm was implemented, it took a period of approximately 1 1/2 man months for the development of the colliding beam grammar, the writing and debugging of the primitive recognizers, and the performance of the analysis. The reasons for this relatively short period of time are:
   (a) Each primitive recognizer could be treated almost independently of the others due to the SPDL environment in which it is used.
   (b) Analysis by parsing allows efficient debugging. The program flow can be immediately obtained from the parsing stack. The value and failure lists indicate exactly what has been found and where.
   (c) The visual display of the parse, especially during backtracking, enabled fast on-line detection of bugs.
2. It was possible to use crude primitive recognizers in our system primarily because of its goal-directed nature, but also because most of the primitives were fairly isolated.
3. Exclusive of input-output time, each picture analysis was completed in approximately 7 seconds on the IBM 360/50. This time is comparable to other hand-coded systems and just indicates the fact that regardless of the analysis method, most of the time is spent in accessing single digitisings.

CONCLUSIONS

We have described a syntax-directed picture analysis system based on the use of a formal picture description scheme. The system accepts a description of a class of pictures in terms of a grammar generating sentences in the PDL picture language; the grammar is explicitly used to direct the analysis or parse, and to control the calls on pattern classification routines for the primitive picture components. In addition to providing a general framework in which a large class of pictures may be described and hopefully analyzed, our approach automatically incorporates a form of contextual pattern recognition that, in many instances, should assist and simplify the classification of primitive patterns. The potential benefits of our methods were demonstrated by the application of an implemented system to the analysis of a small sample of spark chamber film; these benefits include ease of implementation and modification, with little sacrifice in machine efficiency.

This research points to a number of interesting problems and extensions for future work. PDL has some obvious limitations; prominent among these are the inability to conveniently express more complex relations such as "inside of" or "overlapping", and the restriction to only two points of concatenation for a primitive. Can the language be generalized to handle the above without destroying its essential simplicity? PDL is a description language; it would also be useful to have a language in which picture processing systems may be implemented,

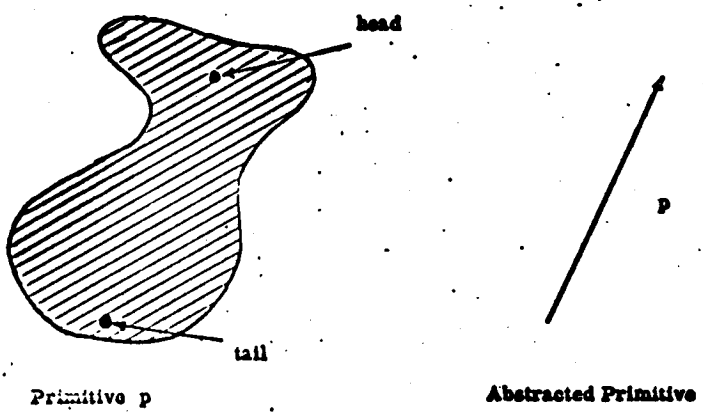


Figure 1.

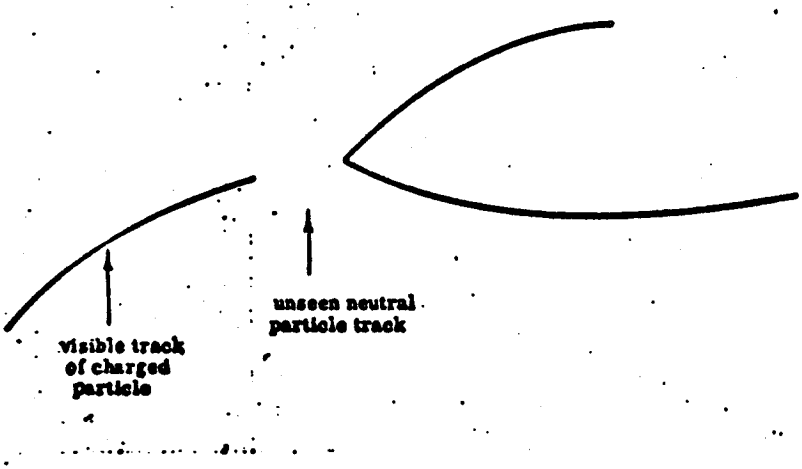

unseen neutral particle track
visible track of charged particle

Figure 2 (a).

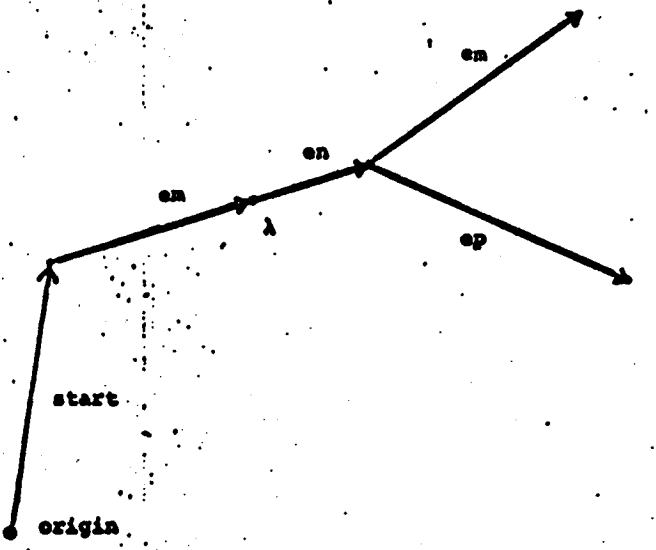


Figure 2 (b).

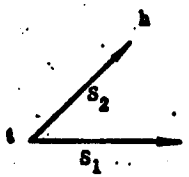
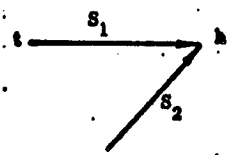
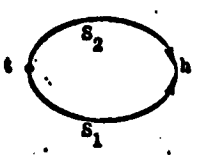
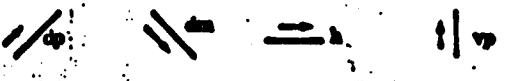
$T_s(\alpha)$	Graph
$(S_1 + S_2)$	
$(S_1 \times S_2)$	
$(S_1 - S_2)$	
$(S_1 * S_2)$	

Figure 3.



Primitive Classes

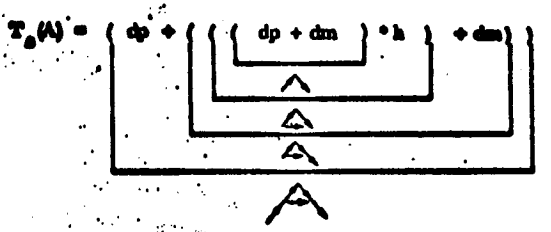
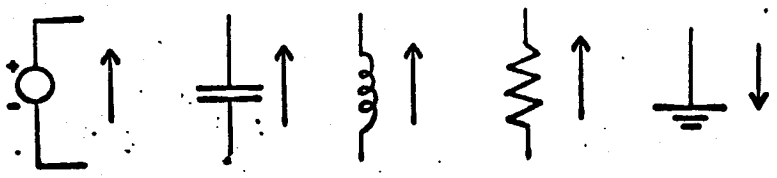
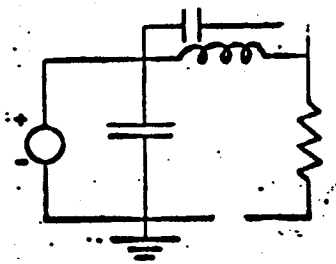


Figure 4 (a).



Primitive Classes

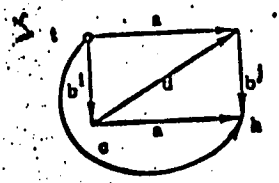


{(V*(C*((L*C)+R)))*G}

Circuit

PDL Description

Figure 4 (b).



$T_0(\alpha) = ((b^1 \cdot a) \cdot ((((/b^1) \cdot d) \cdot (/b^1))) \cdot ((a \cdot b^1) \cdot d))$

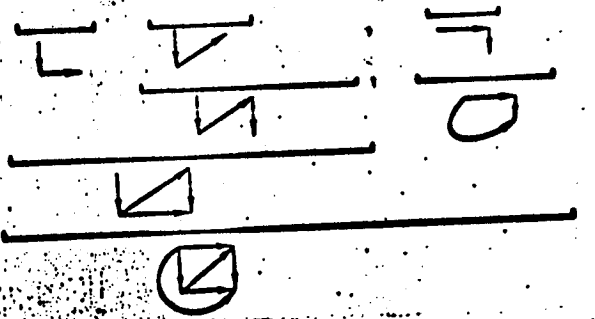


Figure 5.

$\mathcal{L}_1$

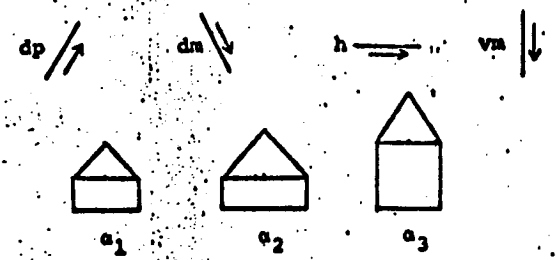
$P \rightarrow A \mid HOUSE$

$A \rightarrow (dp \cdot (TRIANGLE \cdot dm))$

$HOUSE \rightarrow ((vm \cdot (h \cdot (\sim vm))) * TRIANGLE)$

$TRIANGLE \rightarrow ((dp \cdot dm) * h)$

$\mathcal{L}(\mathcal{G}) = \{(dp \cdot (((dp \cdot dm) * h) \cdot dm)),$
$((vm \cdot (h \cdot (\sim vm))) * ((dp \cdot dm) * h))\}$



$T_a(a_1) = ((vm \cdot (h \cdot (\sim vm))) * ((dp \cdot dm) * h))$

$K_a(a_1)$:

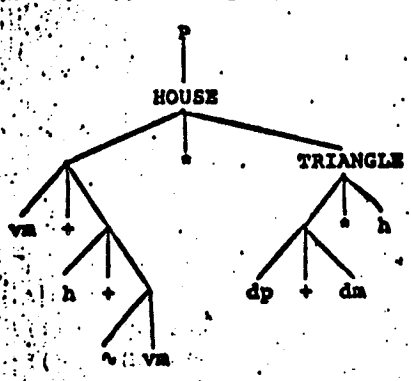
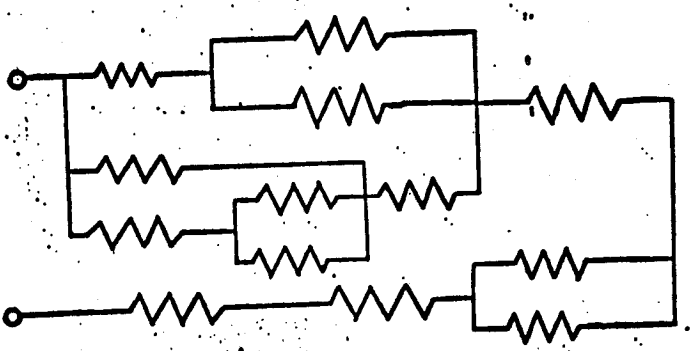


Figure 6.

Representative Primitives



Example

RN → (b + (CCT + b))
CCT → r|SER|PAR
SER → (r + CCT)|(PAR + CCT)
PAR → (r * CCT)|(SER * CCT)

The Grammar

Figure 7.

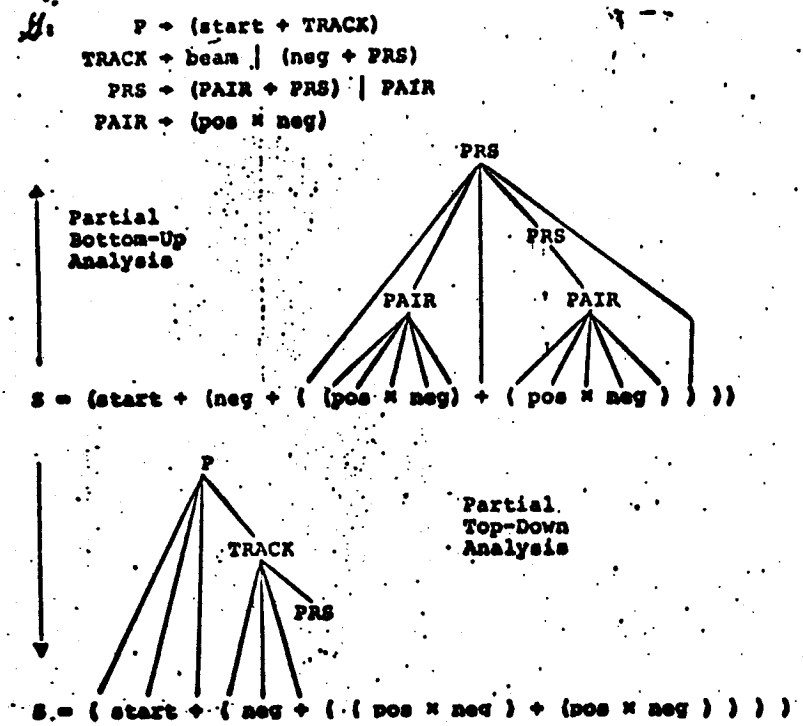

P → (start + TRACK)
TRACK → beam | (neg + PRS)
PRS → (PAIR + PRS) | PAIR
PAIR → (pos × neg)
PRS
PRS
PAIR
PAIR
Partial
Bottom-Up
Analysis
S = (start + (neg + ((pos × neg) + (pos × neg))))
P
TRACK
PRS
Partial
Top-Down
Analysis
S = (start + (neg + ((pos × neg) + (pos × neg))))

Figure 8 (a).

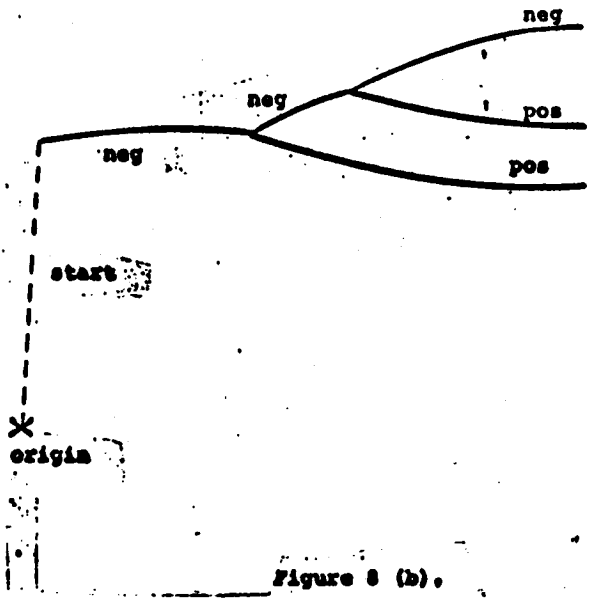


Figure 8 (b).

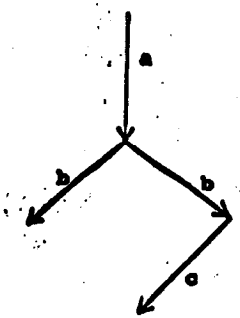


Figure 9 (a).

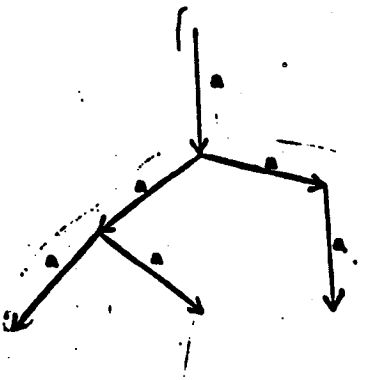


Figure 9 (b)

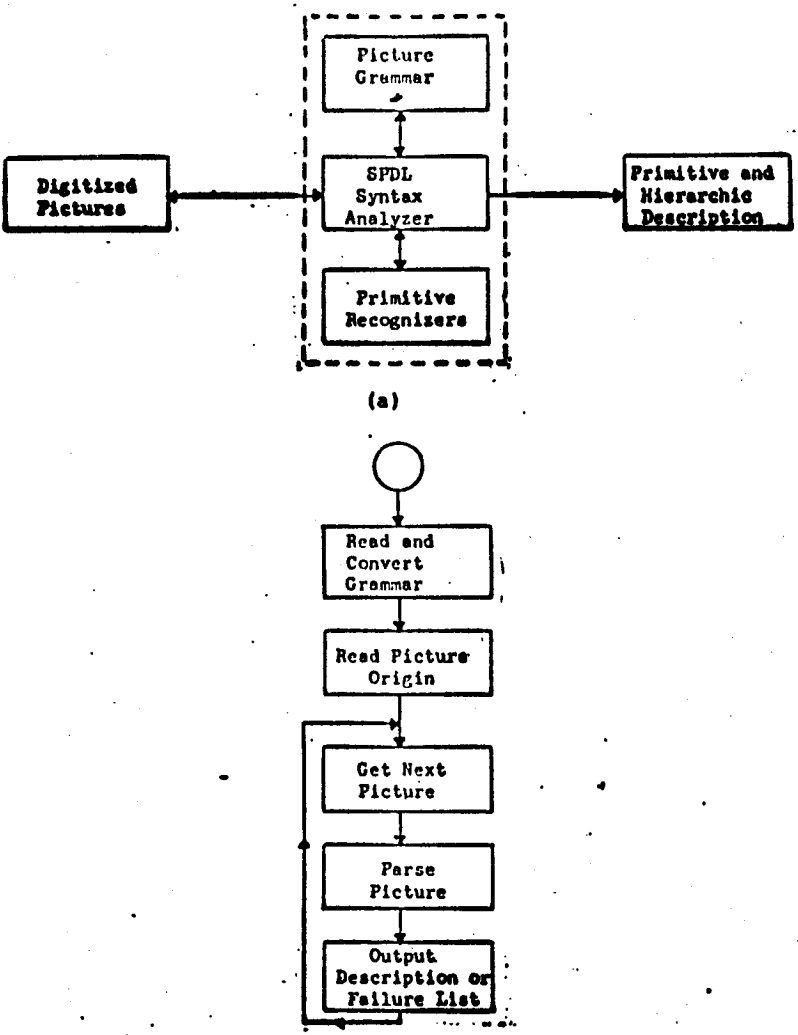

Picture Grammar
Digitized Pictures
SPDL Syntax Analyzer
Primitive and Hierarchic Description
Primitive Recognizers
(a)
Read and Convert Grammar
Read Picture Origin
Get Next Picture
Parse Picture
Output Description or Failure List
(b)

Figure 10.

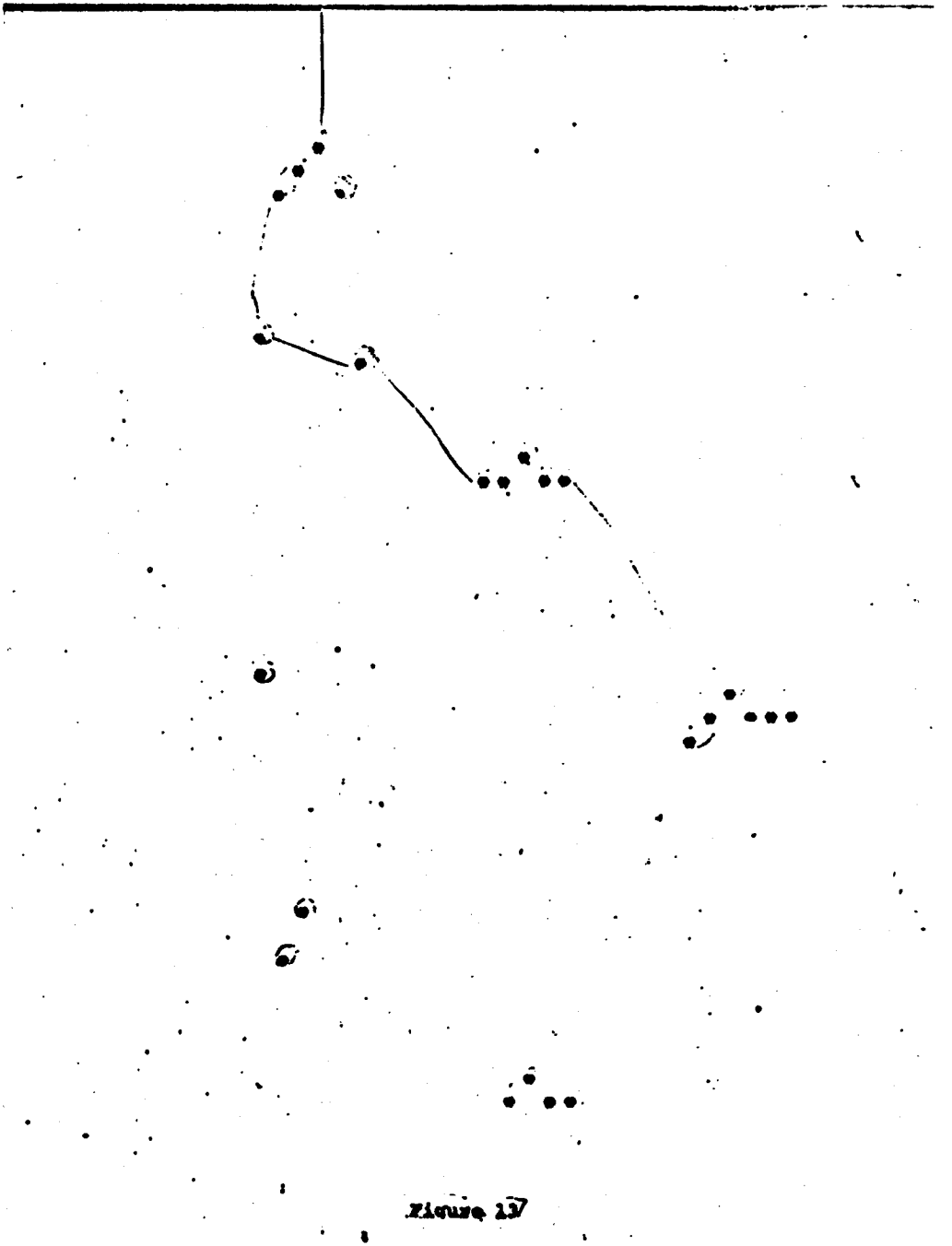


Figure 13

A. C. SHAW

Department of Computer Science
Cornell University

Technical Report
No. 69-34

April 1969

ERRATA

1. P. 8, line 4, add "}" at end of line
   line 14, read "generically" for "generaically"

2. P. 24, line -10, read "$(T_{\sim d} \times T_{\lambda})$" for "$T_{\sim d} \times T_{\lambda}$"

3. P. 27, line 6, read "Con[s]" for "Cons[s]"

4. P. 31, line -3, read "else" for "else"

5. P. 33, line -7, read "[D]" for "(D)"

6. P. 40, line -8, read "their" for "this"

7. P. 47, line -1, read "primitive" for "primitives"

8. P. 51, line 17, read "83-88" for "83-88"

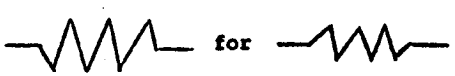
9. FIGURE TITLES, line 13, read "resistance" for "resistance"

10. Fig. 4(b), read "((V*(C*(R+(L*C)))) × G)"
    for "((V*(C*((L*C) + R))) × G)"

11. Fig. 7, (1) read "→" for "+" above primitive b

    (2) read "RN→(b+(CCT+(∼b)))"
    for "RN→(b+(CCT+b))"

    (3) In last row of "Example" read



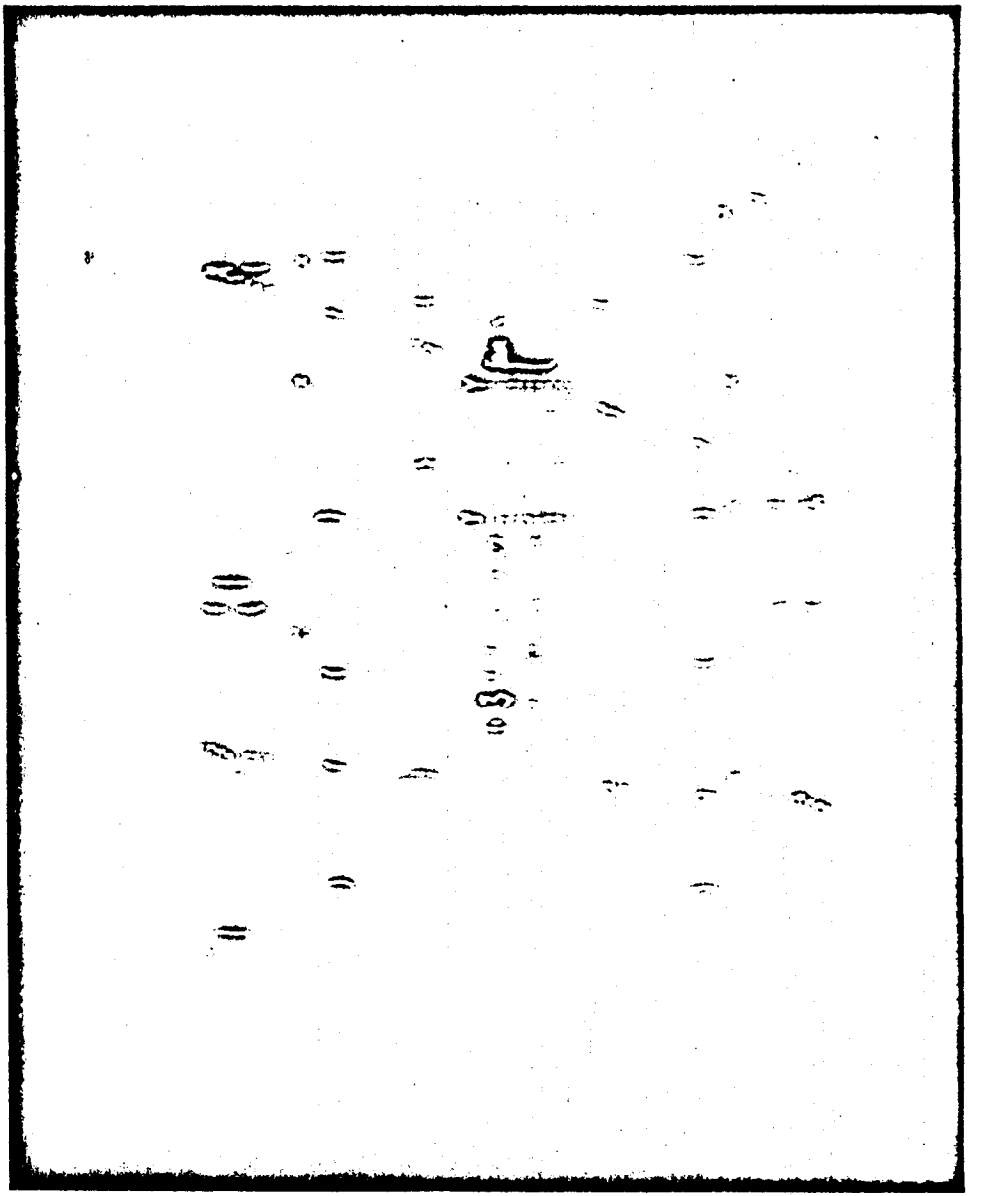


Figure 12.

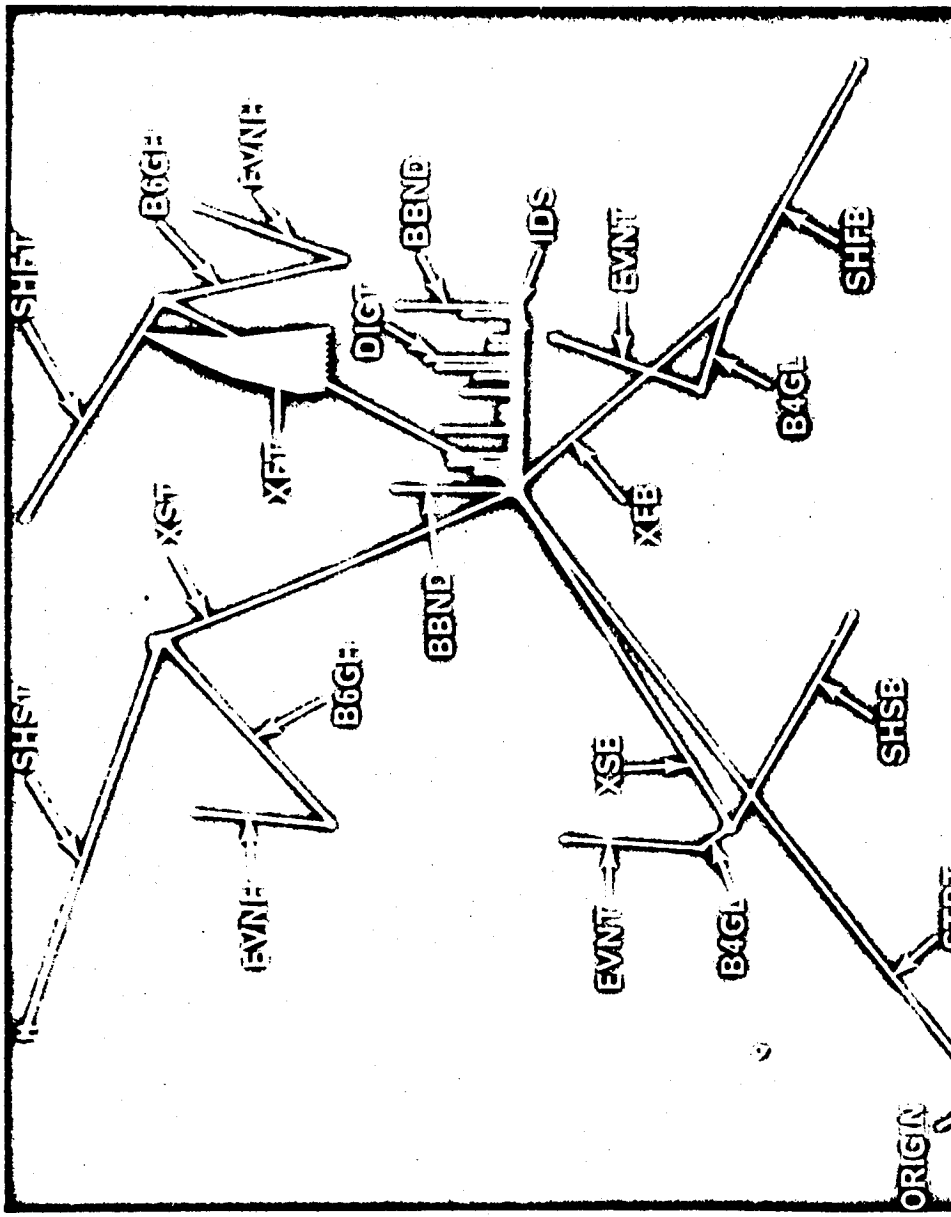

B6GL
EVNT
BBND
IDS
EVNT
SHFB
B4GL
XFB
XST
BBND
B6GL
SHSB
XSE
SHSE
EVNT
B4GL
ORIGIN

Figure 14.


START