SLAC-84
UC-32, Mathematics and Computers

THE FORMAL DESCRIPTION AND PARSING OF PICTURES

April 1968

by

Alan C. Shaw

Technical Report
Prepared Under
Contract AT(04-3)-515
for the USAEC
San Francisco Operations Office

ACKNOWLEDGMENTS

I am deeply indebted to my thesis advisor, Professor William F. Miller, for the many valuable discussions we have had, for his constant encouragement and enthusiasm, and for providing a stimulating atmosphere at SLAC that made it not only possible but a pleasure to do research. I am grateful to Professors G. Forsythe, D. Gries, and R. Reddy for their constructive readings of the thesis.

No research is accomplished in a vacuum. I wish to thank the members of the SLAC Computation Group, the SLAC Automatic Data Analysis Group, and the SLAC-IBM Graphics Study Group for our useful talks on the practical problems arising in film data analysis and for their cooperation in making available some basic systems programs; Charles Dickens of SLAC was particularly helpful. The critical evaluation of parts of this work by my colleagues, Ira Pohl and Donald Kaplan, is gratefully acknowledged.

I wish to express my appreciation to Sandra Hatch for her excellent typing and preparation of this report, and to Dorothy McGrath for the fine illustrations.

I dedicate this work to my wife Elizabeth.

This research was supported in part by The Atomic Energy Commission, the SLAC-IBM Graphics Study Project, and The National Science Foundation grant GP-7615.

TABLE OF CONTENTS

CONTENTS (continued)

CONTENTS (continued)

TABLES

FIGURES

FIGURES (continued)

FIGURES (continued)

FIGURES (continued)

FIGURES (continued)

FIGURES IN APPENDIX

CHAPTER 1

INTRODUCTION

This thesis is concerned with the analysis and generation of pictures by computer. The name "picture processing" will be used to describe this subject area. While there is occasional reference to generation, the main emphasis is on the more difficult problem of analysis. The latter has been traditionally called pattern recognition or classification; here, analysis is interpreted more generally to mean the derivation of picture descriptions.

A distinction between "natural" and "artificial" pictures can be made analogous to that between natural and artificial languages. Artificial languages and subsets of natural languages can be effectively analyzed using formal methods for expressing and dealing with syntax and semantics. A similar approach has been developed here for the analysis of suitably restricted classes of pictures. Any picture whose component connectivity can be meaningfully described by a graph is a candidate for the system. This will be the operational definition of an artificial picture.

Despite the introduction of a modest amount of theoretical material, the results of this thesis are essentially pragmatic. The direct applicability of this approach to <u>real</u> picture processing problems is demonstrated. In most cases, the discussion assumes that the pictures have been converted to machine-readable form by digitization hardware.

1.1 HISTORY OF PROJECT AND CONTRIBUTIONS

This research was started seriously during the spring of 1966. The advantages of representing pictures by graphs and using graph properties as an aid to their analysis were demonstrated by Clark and Miller [1966] in their particle physics film recognition system; a subsequent suggestion by Professor Miller that the model of graph theory might prove generally useful in picture processing was the original inspiration of this research. The developments were further influenced by the work of Narasimhan [1962-1966] in suggesting and applying linguistic methods for picture analysis and generation, and that of Kirsch [1964] in defining a number of outstanding problems resulting from a language interpretation of pictures. Finally, the complexity of present systems and the author's early painful experiences in programming some picture manipulation routines confirmed the need for "a better way".

A general picture processing model was first postulated; the basic requirement within this model was a formal picture description scheme. To this end, the PDL picture description language was conceived, developed, and applied during the fall of 1966 and winter of 1967 (Shaw [1967a]); at the same time, the ideas and algorithms for parsing (analyzing) pictures were worked out. Research into methods for recognition of specific classes of elementary picture components continued in parallel with the above. An implementation of an analysis system was completed in the spring and the latter was applied to some spark chamber photographs during the summer of 1967. Concurrently, preliminary work was begun on the development of an interactive generation system based on the PDL language (Noyelle [1967], George [1967]). The PDL language was found to be particularly well-suited as the notation for a "picture calculus"; the foundations of this calculus were laid in reports by Miller and Shaw [1967a, b], and some basic theorems were derived in Shaw [1967b].

The original work and contributions of the author include:

1. a simple and general picture processing model,
2. a formal picture description scheme which allows the description of a large class of pictures in terms of their primitive elements, the relationships among primitives, and the meaningful structures formed by sets of primitives, and
3. the concept and development of picture parsing--the description-directed analysis of pictures.

In addition, an implementation and analysis has demonstrated the validity of this approach; a simple recognizer for a variety of line-like elements was developed for the application. In contrast to other methods, this approach offers the advantages of simplicity and generality for both description and analysis.

1.2 ORGANIZATION

The thesis is organized in a "top-down" manner, proceeding roughly from the general to the specific. The second chapter motivates and describes the picture processing model; related work is surveyed within this model. The details of the picture description scheme are presented in Chapter 3, including some of its formal properties. The following chapter illustrates its descriptive power and limitations by means of a series of picture description examples. Chapter 5 describes the rationale and algorithms for a goal-oriented picture parser; the implemented system is presented. The next chapter is devoted to the construction of primitive recognizers or pattern recognition routines for blobs, line segments, and blank primitives. The implemented system is applied to the analysis of spark chamber film and the results are given in Chapter 7. The final chapter lists a number of open problems and summarizes the major advantages of this approach.

Algorithms and the meaning of some constructs are conveniently defined by recursive functions consisting of conditional forms and ALGOL-like statements, with symbol lists as data and arguments.

CHAPTER 2

A LINGUISTIC MODEL FOR PICTURE PROCESSING

The term "model" denotes the general framework or "paradigm" (Kuhn [1962]) within which workers pose and solve problems. Until recently, most theoretical work in pattern recognition has, either implicitly or explicitly, been based on the receptor/categorizer model (RCM) of Marill and Green [1960]. While many useful and interesting results have been produced, this model, nevertheless, has some very serious limitations which are discussed in the next section.

2.1 THE RECEPTOR/CATEGORIZER MODEL

The analysis of pictures or pattern recognition proceeds as follows: A picture is first reduced to a "feature" set by the receptor; this is a set of quantities which may represent anything from the raw digitized values (or analog signal) at one extreme to the results of a complex feature extraction process on the other. The feature set is then assigned to one of a finite number of classes or patterns by the categorizer. The assignment is the recognized pattern class to which the picture supposedly belongs. It is often convenient to add a "reject" class; when the input cannot be assigned to a known pattern, the categorizer puts it in the reject class.

There has been a tremendous quantity of research accomplished within this model (see the large pattern recognition bibliographies of Minsky [1961] and Shaw [1966a]). Receptor work has included the development and application of techniques for:

1. noise reduction (preprocessing) such as smoothing (Unger [1959]) and local averaging (Selfridge [1955]),
2. extraction of global features, such as moments (Alt [1962], Hu [1962]) and n-tuples (Bledsoe and Browning [1959]),
3. topological feature extraction (Sherman [1959], Rosenfeld and Pfaltz [1966]),
4. curvature point characterization (Zahn [1967], Freeman [1961], and
5. combining features into complex features (Uhr and Vossler [1963]).

Most of the theory has dealt with the problem of categorization or classification. The principal technique is to treat the feature or measurement set as a point in a multidimensional space; the task of the categorizer then becomes one of partitioning the space so that measurements from pictures belonging to the same pattern class are "close" (according to some metric) and measurements from pictures of different classes are far apart. The use and limitations of partitioning by hyperplanes or linear discriminants has been exhaustively treated; some non-linear discriminants, for example, polynomials have also been studied (Sebestyen [1962]). When information about the probability distributions of the measurements in each class is available, methods of statistical decision theory can be employed to partition the space (Chow [1957]). Machine learning has been investigated in both the receptor and categorizer stage; in the former, measurements and weights on measurements have been learned successfully in some situations (Uhr and Vossler [1963]); in the latter, the use of adaptive systems for partitioning the space and for learning probabilities has received much attention (Nilsson [1965]).

The RCM is the basis for many recognition systems, notably in character recognition (BCS [1967]). The model fails to be useful when analyzing complex pictures where the structure and interrelationships among the picture components are the important factors. To illustrate this point in a simple setting, consider the one-dimensional pattern recognition task required of a programming language translator, for example, an ALGOL 60 compiler (Randall and Russell [1964], Shaw [1966b]). One general purpose of the syntax analysis phase of the compiler is to categorize an input program into one of two mutually exclusive classes --the class of syntactically correct programs and its complement. Theoretically, one could envision a receptor which produces a feature vector from an input program; the categorizer then would determine in which of the two possible subspaces the feature vector lies. While this could be done in principle, it is never considered seriously because of the complexities involved; for example, what is the feature set for a program? Even if this approach were practically feasible for program classification, it would not produce the most important byproduct of a successful analysis, i.e., a description of the structure of the input program. Parenthetically, RCM is a recognition or analysis model and it is doubtful whether it would be of any value in picture generation.

Richly-structured pictures that are difficult to analyze within the RCM include those produced by high energy particle physics reactions (Adler et al. [1966]), line drawings (Roberts [1963], text (not isolated characters), and some biomedical pictures (Ledley et al. [1965]). What is required in these examples is a description of the pictures in which the meaningful relations among their subparts are apparent. Unfortunately, there has been no general mechanism for either describing this type of picture or analyzing it; each application has required a "one-of-a-kind" system. In these systems, the appropriate place to apply the RCM is for the recognition of the basic components of the pictures.

2.2 THE LINGUISTIC MODEL

In a series of papers, Narasimhan [1962, 1963b, 1964, 1966] has forcefully stated the case for another approach to pattern recognition:

> Categorization, clearly, is only one aspect of the recognition problem; not the whole of it by any means. It is our contention that the aim of any recognition procedure should not be merely to arrive at a 'Yes', 'No', 'Don't know' decision but to produce a structured description of the input picture. Perhaps a good part of this confusion about aims might have been avoided if, historically, the problem had been posed as not one of pattern *recognition* but of pattern *analysis* and *description*. (Narasimhan [1962]).

This writer is in entire agreement with the above, and, in fact, was largely motivated by these and similar remarks to develop a linguistic model for picture processing. The important phrase in the above quotation is "structured description of the input picture". This thesis consists primarily of an interpretation of this phrase (the picture processing model), the ramifications of this interpretation (the PDL language and picture parsing) and the results of an implementation.

The linguistic model for picture processing is comprised of two parts:

1. a general model within which pictures may be described (i.e., a meta-description formalism), and
2. an approach to the analysis (and generation) of pictures based directly on their descriptions.

The description D of a picture α will consist of two parts--a primitive or terminal symbol description T and a hierarchic description H; this can be written $D(\alpha) = (T(\alpha), H(\alpha))$. T and H, in turn, each have a syntactical or structural component T_s and H_s, and a semantic or value component T_v and H_v . I.e.,

$$T(\alpha) = (T_s(\alpha), T_v(\alpha))$$

$$H(\alpha) = (H_s(\alpha), H_v(\alpha)) .$$

$T_s(\alpha)$ describes the elementary component classes or primitives in α and their relationship to one another; $T_v(\alpha)$ gives the values or meaning of the components of α . It should be noted that the primitives in $T_s(\alpha)$ denote classes; define $\mathcal{P}(T_s)$ as the set of all pictures with primitive structure T_s .

Example 1:

Let ℓ name the set of all straight line segments. Let c name the set of all circles. ℓ and c are picture primitives. Let $\odot$ denote the geometric relationship of intersection.

Then, if a picture α contains a line segment intersecting a circle, $T_s(\alpha) = \ell \odot c$; $T_v(\alpha)$ could be the list (v_ℓ, v_c), where v_ℓ is the pair of endpoint coordinates of ℓ and v_c is the center coordinates and radius of c. $\mathcal{P}(\ell \odot c)$ is the set of all pictures consisting of a line segment intersecting a circle.

Consider a set of rules or grammar $\mathcal{G}$ generating a language $\mathcal{L}(\mathcal{G})$ whose "sentences" are primitive structural descriptions. Then, $\mathcal{G}$ is said to describe the picture class $\mathcal{P}_{\mathcal{G}} = \bigcup_{T_s \in \mathcal{L}(\mathcal{G})} \mathcal{P}(T_s)$. For a given picture $\alpha \in \mathcal{P}_{\mathcal{G}}$, $H_s(\alpha)$ is the ordered set of rules of $\mathcal{G}$ that were used to generate $T_s(\alpha)$; that is, $H_s(\alpha)$ is the "linguistic" structure or parse of $T_s(\alpha)$ according to $\mathcal{G}$. A one-to-one correspondence exists between the elements of a set of semantic or interpretation rules $\mathcal{S}$ and the elements of $\mathcal{G}$. $H_v(\alpha)$ is defined as the result of obeying the corresponding semantic rule for each rule of $\mathcal{G}$ used in $H_s(\alpha)$.

<u>Example 2</u>:

Let $\mathcal{G}$ be the phrase structure grammar (Chomsky [1957]): $\mathcal{G} = \{LC \rightarrow L,\ LC \rightarrow C,\ LC \rightarrow L \odot C,\ L \rightarrow \ell,\ C \rightarrow c\}$, where ℓ, c, and $\odot$ are defined as in Example 1. Let $\mathcal{S} = \{v_{LC} := v_L,\ v_{LC} := v_C,\ v_{LC} := \text{xsect}(v_L, v_C),\ v_L := v_\ell,\ v_C := v_c\}$, where v_i, $i \in \{L, C, LC\}$, is the value of the corresponding grammar rule and xsect is a function that computes the intersection(s) of a line with a circle. Then, $\mathcal{L}(\mathcal{G}) = \{\ell, c, \ell \odot c\}$ and $\mathcal{P}_{\mathcal{G}} = \mathcal{P}(\ell) \cup \mathcal{P}(c) \cup \mathcal{P}(\ell \odot c)$. If $T_s(\alpha) = \ell \odot c$ for a given $\alpha \in \mathcal{P}_{\mathcal{G}}$, $H_s(\alpha)$ could be the simple tree:

$$\begin{tikzcd}
 & LC \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\
L \arrow[d, no head] & \odot & C \arrow[d, no head] \\
\ell & & c
\end{tikzcd}$$

$H_v(\alpha)$ could be the list structure:

$$\begin{tikzcd}[column sep=small]
\{v_{LC} & , \arrow[r, bend left] \arrow[rrrr, bend right] & v_L & , \arrow[r, bend left] & v_\ell , & v_c & , \arrow[r, bend left] & v_c\}
\end{tikzcd}$$

Several features of the description model require emphasis. It is important to note that the grammar must be capable of generating primitive structural descriptions of <u>all</u> pictures being considered. No restrictions are made on the form of any of the components of D . A final point is the essential difference between primitive and hierarchic descriptions; the "meaning" of a picture is expressed by both. Thus, several grammars may be used to generate the same class of primitive descriptions, but the hierarchic description of a picture and hence its meaning may be different for different grammars. Even more generally, the same picture class may be described by totally different primitive and hierarchic descriptions; the intended interpretation of the picture dictates its description.

With the description model, the solution to the picture analysis problem can now be formulated:

1. The elementary components or primitives which may appear in a class of pictures are named and given a meaning.
2. The picture class is described by a generative grammar $\mathscr{G}$ and associated semantics $\mathscr{S}$.
3. A given picture α is then analyzed by parsing it according to $\mathscr{G}$ and $\mathscr{S}$ to obtain its description $D(\alpha)$; that is, $\mathscr{G}$ and $\mathscr{S}$ are used explicitly to <u>direct</u> the analysis.

Conversely, picture generation can be viewed as the execution of descriptions.

Descriptions are then not only the desired results of an analysis, but they also define the algorithms that guide the recognition. The advantages of this approach are:

1. It defines a general strategy for analyzing pictures; this implies that a general-purpose analysis program may be written for a formally defined description scheme.
2. The design of the recognizers for the primitive components of a picture is simplified. Each recognizer may be defined independently of the others and thus may include its own preprocessing. In addition, directed recognition can be done more easily than global searches for all primitive components.
3. Picture processing systems based on this model can be implemented quickly and reliably.

As the structural complexity of pictures increases, the value of this description and analysis model becomes greater. At the lowest level, a picture is described by one primitive component; this may be analyzed using the RCM model. Each of the elementary components of more complicated pictures can also be recognized within the RCM model. Thus, the RCM model is included as a part or subset of the picture processing model.

Much of the power of the model lies in the flexibility in the hierarchic description. If the purpose of an analysis is pattern classification, then the classification can be inserted directly in $\mathcal{L}$ or $\mathcal{J}$ so that the resulting description explicitly contains the pattern class. $\mathcal{L}(\mathcal{G})$ could include a description of the complement of all well-formed pictures; in this case, pictures that cannot be properly classified may be described as such.

It should be mentioned that syntax-directed translation of programming languages (Irons [1961], Floyd [1964], Feldman and Gries [1967], Shaw [1966b]) can be interpreted as the analysis of patterns of linear strings and thus put within the model. In this case, the primitive description is obtained immediately--the input program corresponds to T_s and the meaning of the basic symbols of the language to T_v . The grammar $\mathcal{G}$ is generally a BNF grammar plus some constraints on the use of identifiers; the semantics $\mathcal{S}$ is most often a set of code-generating rules. The analysis of a well-formed program yields the syntactic structure of the program and an equivalent program in some other language. The similarity of the picture processing model to that used in translator writing is no accident; many of the ideas and results of language theory and compiler construction are used in later parts of this thesis. The name "linguistic model" is derived in part from the above considerations (Narasimhan [1962] first used this name as applied to picture processing).

The next section surveys briefly those research efforts that are related to this one. The model is used as a common framework for the discussion. The purpose of the survey is to allow later comparisons, to assign credit for some of the early research that has influenced the present work, to illustrate the generality of the model, and to enable the reader to put this work in its proper perspective.

2.3 THE LINGUISTIC APPROACH TO PICTURE PROCESSING: A BRIEF SURVEY

The literature survey of Feder [1966] covers the few basic developments up to and including 1965; since then, there has been a relative surge of activity.

There are several early works that explicitly utilized primitive descriptions. Grimsdale et. al. [1958] produced geometric descriptions of hand-drawn line figures, such as alphabetic characters; the description consisted of an encoded list of the picture curves, their connectivity, and geometric properties. Sherman [1959] reduced a hand-printed letter to a graph, and then built a character description out of the topological and geometric features of the abstracted picture. Neither T_s nor T_v is defined formally in the above examples; picture analysis (recognition) occurs by comparing or matching picture descriptions with descriptions of standard patterns. Eden [1961, 1962] presented a formal system for describing handwriting. The primitive elements are a set of basic "strokes" or curves; the value of each stroke is a point pair (the end-points) and a direction. Eden gave a set of rules $\mathscr{L}$ for concatenating or collating strokes to form letters and words. The description T_s of a word of handwriting is then a sequence of n-tuples of strokes, each n-tuple representing a letter. This is one of the first works where the author recognizes the benefits of a generative description:

> Identification by a generative procedure leads to a clear definition of the set of permissible patterns. The class of accepted patterns is simply the set which can be generated by the rules operating on the primitive symbols of the theory. (Eden [1962]).

Unfortunately, Eden did not report any attempts at using his scheme for recognition purposes--perhaps, because of the complexities of actually trying to recognize handwriting; however, his descriptions were used for generation.

The pioneering work in suggesting and applying a linguistic model for the solution of non-trivial problems in picture processing was done by Narasimhan [1962, 1963a, b, 1964, 1966]. He first proposed a general linguistic approach in 1962, calling it a "linguistic model for patterns"; he has since applied it to the analysis of bubble chamber photographs using a parallel computer [1963a, 1964, 1966], and to the generation of "hand-printed" English characters [1966]. Narasimhan restricts his model to the class of pictures containing only thin line-like elements. T_s is a list of the "basic sets" and their connectivity, where basic sets refer to neighborhoods on the picture having specified topological properties, for example, the neighborhood about the junction of two lines or the neighborhood about an endpoint of a line. Two sets are said to be connected if there exists a "road" or line-like element between them. T_v is the value of the sets (their topological meaning) and the geometry of the connecting roads. A set of rules or grammar $\mathcal{G}$ then describes how strings of connected sets may be combined into other strings and phrases; phrases are of the form: $\langle\text{name}\rangle(\langle\text{vertex list}\rangle)$, for example, ST(1, 2, 3), where the $\langle\text{vertex list}\rangle$ labels those points that may be linked to other phrases. Finally, there are additional rules of $\mathcal{G}$ for conbining phrases into sentences. The description H_s of a picture is then a list of sentences. Analysis proceeds from the "bottom up", first labeling all points as basic sets or roads, then forming phrases, and last of all, sentences. Narasimhan does not define a general form for $\mathcal{G}$ and D . In the bubble chamber application, $\mathcal{G}$ is implicitly defined in the program itself. On the other hand, the generation of English "hand-printed" characters is explicitly directed by a finite-state generative grammar $\mathcal{G}$ and an attribute list $\mathcal{A}$, the latter specifying some geometric properties of the characters, for example position, length, and thickness. The primitives are simple geometric forms, such as straight lines or arcs; the definition of each primitive includes a set of labeled vertices to which other primitives may be attached. Productions or rewriting rules in $\mathcal{G}$ are of the form:

$$S(n_s) \rightarrow S_1 \cdot S_2(n_{S_1S_2}; n_{S_1S}; n_{S_2S}),$$

where S_1 is a terminal symbol (primitive name) or non-terminal symbol (phrase name), S_2 is a terminal symbol, S is a non-terminal symbol--the defined phrase--, $n_{S_1S_2}$ is a list of the nodes of concatenation between S_1 and S_2, n_{S_1S} and n_{S_2S} define the correspondence between the nodes of S_1 and S_2 and those of S, and n_S is a node list labeling the nodes of S. Figure 2.1 illustrates Narasimhan's rewriting rules for generating the letter "P", the primitives required, and the generated letters. In the above implementations, all nodes of possible concatenation must appear in the description; this is cumbersome for simple pictures such as the English alphabet, and might be unmanageable for more complex pictures. The system can only describe connected pictures and some other mechanism is required when dealing with pictures whose subparts are not connected.

$$\begin{aligned} PE(1, 2, 3) \rightarrow\ & v \cdot d'(11, 23; 2, 3; 2) \,\Big|\, \\ & r \cdot d'(11, 23; 2, 3; 2) \end{aligned}$$

$$P \rightarrow PE$$

Figure 2.1(a) Rewriting Rules

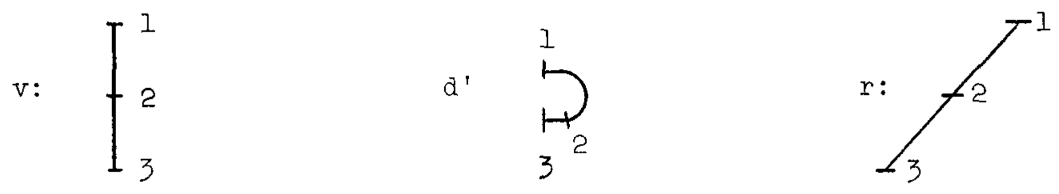


Figure 2.1(b) Primitives

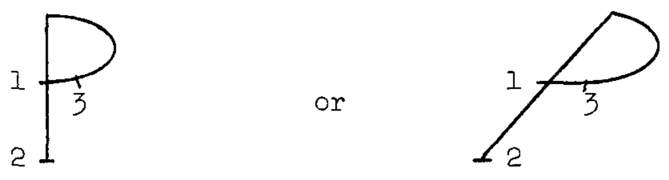


Figure 2.1(c) P and PE

Figure 2.1 Narasimhan's Generation of the Letter "P"

Kirsch [1964], in a stimulating article, argues that the proper way to view picture analysis is within a linguistic framework. Following this line of thought, he posed several problems: How does one

1. express picture syntax or structure,
2. generalize the idea of concatenation to several dimensions,
3. describe geometric relations among picture components,
4. do syntax analysis of pictures, and
5. define picture primitives?

Kirsch gives a two-dimensional context-dependent grammar for 45° right triangles generated in a plane divided into unit squares; this is suggested as an illustration of the possible form of picture grammars. Figure 2.2 contains a sample production and a derived triangle. Here, T_s is a 2-dimensional 45^{o} right triangle with labeled unit squares (the primitives); T_v is the meaning of the labels. There is no semantic portion $\mathcal{J}$ corresponding to the grammar. As Kirsch admits, it is not evident how this approach may be generalized for other pictures. It is also a debatable point whether context-sensitive grammars are desirable since the analysis would be extremely complex. Lipkin, Watt, and Kirsch [1966] argue persuasively for an "iconic" (image-like) grammar to be used for the analysis and synthesis of biological images within a large interactive computer system; however, the authors state that "we cannot at this time show examples of any except the most primitive form of picture grammar." This thesis offers a solution to some of the picture description problems posed by Kirsch for the class of pictures defined in Chapter 1.

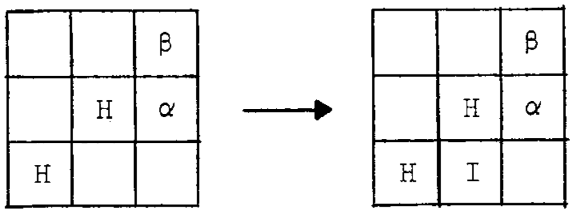


Figure 2.2(a) Sample Production: $\alpha \in \{L, I\}$, $\beta \in \{H, W\}$

				W
			H	L
		H	I	L
	H	I	I	L
V	B	B	B	R

Figure 2.2(b) A Derived Triangle

Figure 2.2 Kirsch's Right Triangle Description

Ledley [1962] and Ledley et. al. [1965] employed a standard BNF grammar to define picture classes. Their published method for the analysis of chromosomes [1965] illustrates this approach. Here, Ledley's "syntax-directed pattern recognition" is embedded in a large picture processing system that searches a picture for objects, recognizes the primitives of an object, performs a syntax analysis of the object description, and finally computes further classifications and some statistics on all the chromosomes found. The object primitives consist of five types of curves from which chromosome boundaries can be generated. An edge-following program traces the boundary of an object in the picture and classifies each boundary segment into one of the primitive classes; since the boundary is a closed curve, a linear string or ordered list of its segment types is sufficient for the description T_s . If T_s represents a chromosome, the parse H_s will contain a categorization of it as, for example, submedian or telocentric type; otherwise the parse fails, indicating the original object was not a chromosome. Figure 2.3 contains samples from chromosome syntax, examples of the basic curve types, and some chromosome descriptions. Ledley's work is an example of a direct application of artificial language analysis methods to picture classification. It is difficult to generalize this approach to figures other than closed curves unless relational operators are included as part of T_s; in the latter case, the most difficult task is obtaining T_s, not parsing the resulting string.

```
                    ⟨arm⟩ ::= B⟨arm⟩|⟨arm⟩ B | A
                   ⟨side⟩ ::= B⟨side⟩|⟨side⟩ B | B | D
⟨submedian chromosome⟩ ::= ⟨arm pair⟩⟨arm pair⟩
```

Figure 2.3(a) Sample Productions

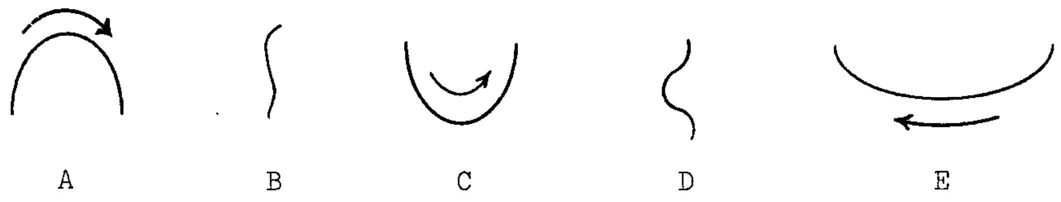
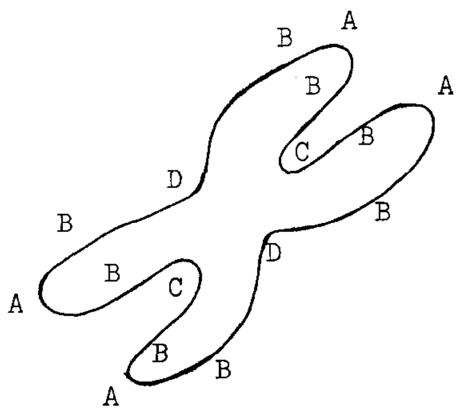
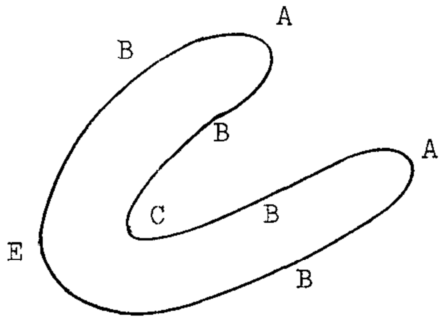


Figure 2.3(b) Basic Curve Types



BCBABDBABCBABDBA

Submedian



BCBABEBA

Telocentric

Figure 2.3(c) Chromosome Examples

Figure 2.3 Ledley's Chromosome Description

Clark and Miller [1966] use the language of graph theory to describe spark linkages and the topology of physics "events" appearing in spark chamber film. These descriptions are embodied in computer programs that apply some graph theorems to assist in the decision-making process and perform the film analysis. The primitive elements of the pictures are sparks; a multi-list structure provides the description T_s and T_v of the spark connectivities. Hierarchic descriptions result from combining sparks according to their geometric and graph properties to form tracks and events. While an explicit linguistic approach is not employed, the underlying graph model acts as a formal description language.

Very recently, Anderson [1967] and Clowes [1967 a, b] have reported some interesting research on the application of linguistic models to picture processing. In Clowes [1967a], a set $\mathcal{S}$ of Boolean functions on pictures is used to define the syntactical classes for hand-written numerals; the successive execution of these functions from the bottom up serves to analyze and describe the pictures. Another approach, based on Chomsky's model for natural language syntax (Chomsky [1965]) was proposed by Clowes [1967b]. Until more experimentation is done, it is not clear how useful this model will be. Anderson [1967] syntactically analyzes pictures *after* their primitive elements have been characterized by conventional pattern recognition techniques. The value of a primitive is its name and 6 positional coordinates: X_{min}, X_{center}, X_{max}, Y_{min}, Y_{center}, Y_{max} . Each syntax rule consists of four structural parts (elements of $\mathcal{S}$) and one semantic part (element of $\mathcal{I}$) . Figure 2.4 contains a syntax rule used in the recognition of hand-printed two-dimensional mathematical notation. The meaning of the notation is as follows:

S_i : the i^{th} element (left to right order) of the right part of the syntax rule.

P_i : a partitioning predicate that S_i must satisfy. C_{ij} is the j^{th} positional coordinate of S_i .

R : a predicate testing the spatial relationship among successfully parsed elements of the right part of the syntax rule.

C_i : the six coordinates to be assigned to the left part of the syntax rule in a successful parse.

M : the semantic rule indicating an action to be taken or the meaning to be given to the rule.

A top-down goal-directed method is used for analysis; the basic idea is to use the syntax directly to partition the picture space into syntactical units. Anderson has described several non-trivial classes of pictures, such as hand-written two-dimensional mathematical formulas, directed graphs, and flow charts; but, as of this writing, has tested the analysis only on simulated "hand-printed" data. The set partitioning strategy is inherently inefficient and there remains the question of what increases in efficiency can be obtained by the various devices he proposes. One of the virtues of this model is the use of predicates which allow the expression of complex relations among the picture parts.

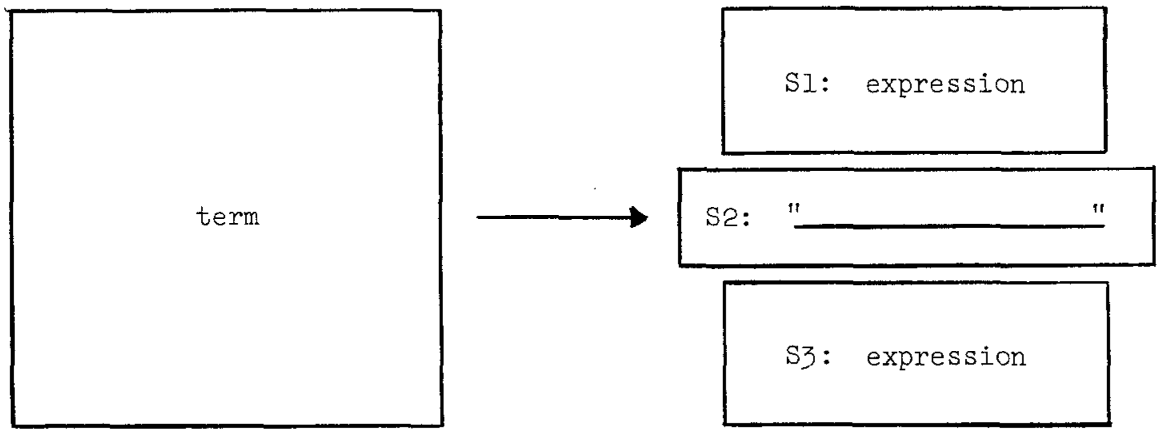


Figure 2.4(a) Graphical Form of Replacement Rule

term $\longrightarrow$

S1:	expression	P1:	$c_{01} > c_{21}$ and $c_{03} < c_{23}$	C1:	c_{21}
			and $c_{04} > c_{26}$	C2:	c_{22}
S2:	horizline	P2:	$\emptyset$	C3:	c_{23}
S3:	expression	P3:	$c_{01} > c_{21}$ and $c_{03} < c_{23}$	C4:	c_{34}
R:	$\emptyset$		and $c_{06} < c_{24}$	C5:	c_{25}
M:	$(s_1)/(s_3)$			C6:	c_{16}

Figure 2.4(b) Tabular Form of Replacement Rule

Figure 2.4 Example of Anderson's Syntax Rules

Underlying a large number of description schemes is the conceptualization of a picture as a graph; in the above survey, the models of Narasimhan, Sherman, and Clark and Miller are clearly of this type. Other picture processing efforts where a graph representation has proved useful include Breeding [1965] and Guzman [1967].

CHAPTER 3

THE PDL PICTURE DESCRIPTION SYSTEM

3.1 PURPOSE AND REQUIREMENTS OF A PICTURE DESCRIPTION LANGUAGE

Two general applications may be envisioned for a formal picture description scheme:

1. It can be a language of discourse about pictures for humans.
2. As part of the picture processing model outlined in Chapter 2, it provides the basis for analysis and generation of pictures by computers.

The PDL language is the result of trying to meet the following requirements:

1. The language must be capable of describing, both to humans and computers, a large and interesting class of pictures.
2. The structure (syntax) and meaning (semantics) should be contained in a picture description.
3. The basic forms of the language should be simple and natural.
4. Picture descriptions must be generative - that is, a reasonable facsimile of a picture can be generated from its description.
5. The language should be used directly (explicitly) by computer programs to solve analysis and generation problems.
6. There must be general algorithms for number 5 which apply to any picture which can be described.
7. Descriptions should be (almost) independent of coordinate systems and the number of levels of digitization.

8. The language must apply to pictures in two or three dimensions.

An evaluation of PDL in terms of the above requirements is given in the last chapter.

3.2 PICTURE PRIMITIVES

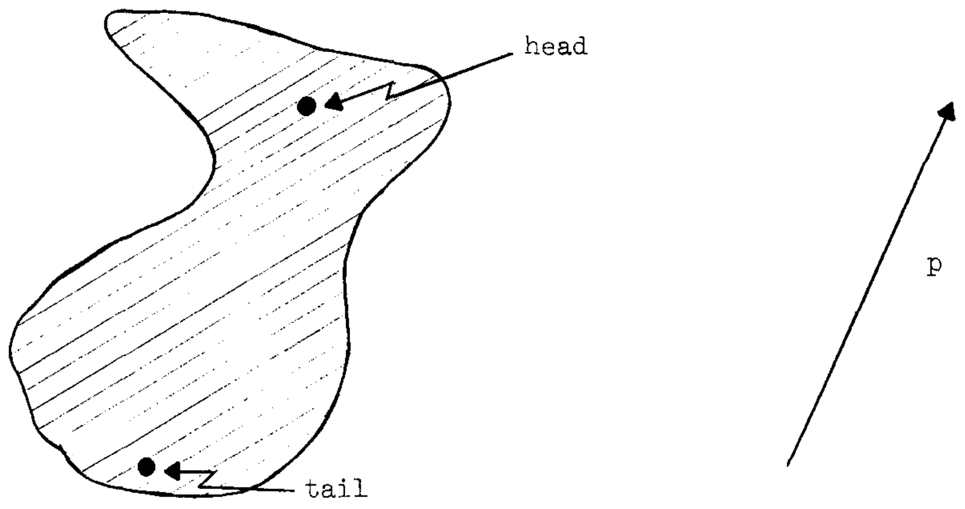
3.2.1 DEFINITION OF A PRIMITIVE

Kirsch [1964] suggests that the elementary or primitive components of a picture be defined as those patterns "which are recognizable by suitable character recognition equipment." The definition is slightly changed here:

> A picture _primitive_ is any picture that can be recognized (generated) by established hardware and software techniques more conveniently than by expressing it (in PDL) in terms of its subparts.

Thus, what constitutes a primitive is a matter of convenience and is dependent on the application and picture class. For example, in character recognition, the characters themselves may be primitives, or it may be more advantageous to consider line and curve segments as primitives and describe the characters in terms of these.

It is required that a primitive have two distinguished points, a _tail_ and a _head_. A primitive can be linked or concatenated to other primitives _only_ at its tail and/or head. Because there are only two points of possible concatenation, a primitive can be represented as a labeled directed edge of a graph, pointing from its tail to its head node (Figure 3.1); this will be a frequent and useful abstraction. Note that generally, there is no inherent direction associated with a primitive pattern per se; the use of directed edges to represent primitives is merely convenient for explaining the description scheme.



Primitive p　　　　　Abstracted Primitive

Figure 3.1 Representation of a Picture Primitive

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 some high energy particle physics reactions (Ford [1963]); 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 3.2). Blank (invis-ilbe) 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 disjoint primitives separated by other patterns, the separating patterns are defined as "don't care" primitives. Blank and "don't care" primitives are therefore allowed.

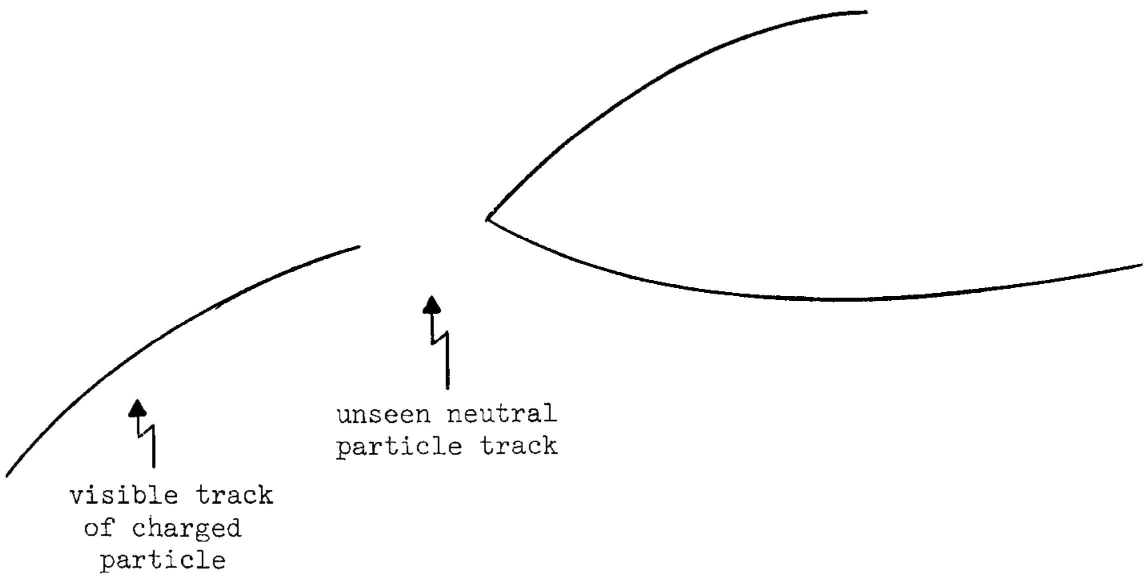

unseen neutral particle track
visible track of charged particle

Figure 3.2 An Invisible Primitive

It is convenient to define one special primitive, the null point primitive λ, having identical tail and head. λ consists only of its tail and head point and will be represented as a labeled node in a graph.

3.2.2 DESCRIPTION OF A PRIMITIVE

A primitive is generally a member of a pattern class; the latter may be described by a name, a tail and head specification, and a recognition (generation) function. The syntax or structure of a primitive is defined as the name of the pattern class to which it belongs; lower case ALGOL ⟨identifier⟩s (Naur et. al. [1963]) will be used to name primitive pattern classes. For a primitive class x, let $\wp(x)$ be the set of all pictures in the class named x (i.e., the class with syntax x); pictures, and thus members of a primitive class, will be designated by the first few letters of the lower case Greek alphabet. The <u>value</u> or <u>semantics</u> of a picture primitive $\alpha \in \wp(x)$, which is contained in a given picture, is defined as the list:

$$\text{value}(\alpha) = (\text{tail}(\alpha), \text{head}(\alpha), v_1, v_2, \dots, v_n),$$

where $\text{tail}(\alpha)$ and $\text{head}(\alpha)$ are the coordinates of the tail and head of α respectively and $v_1, v_2, \dots, v_n$ are an arbitrary number of attributes. The recognition function for $\wp(x)$ is assumed to yield the value of α on success.

The description $D(\alpha)$ of a primitive $\alpha \in \wp(x)$ is the pair:

$$D(\alpha) = (T_s(\alpha), T_v(\alpha)) = (x, \text{value}(\alpha)),$$

where T_s and T_v are defined in section 2.2 of the last chapter. The null point primitive λ has the description:

$$D(\lambda) = (\lambda, (\text{tail}(\lambda), \text{head}(\lambda)))$$

<u>Example</u>:

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

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

3.2.3 THE PRIMITIVE CONNECTIVITY ASSUMPTION

A picture can be represented as a directed graph, where the edges are the abstracted primitives labeled by their primitive class names, some nodes may be labeled λ, and the graph connectivity mirrors the tail/head concatenations of the primitives.

Definition:

A picture is 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.

That this is a reasonable assumption can be seen by considering the extreme case of a picture consisting of a number of disjoint, unrelated primitives. In this case, the geometric relation (coordinates) of each primitive relative to the "origin" of the picture is usually meaningful; the connectivity is obtained by linking the origin to each primitive by blank primitives (Figure 3.4); t_i and h_i point to the tail and head of primitive i in the figure.

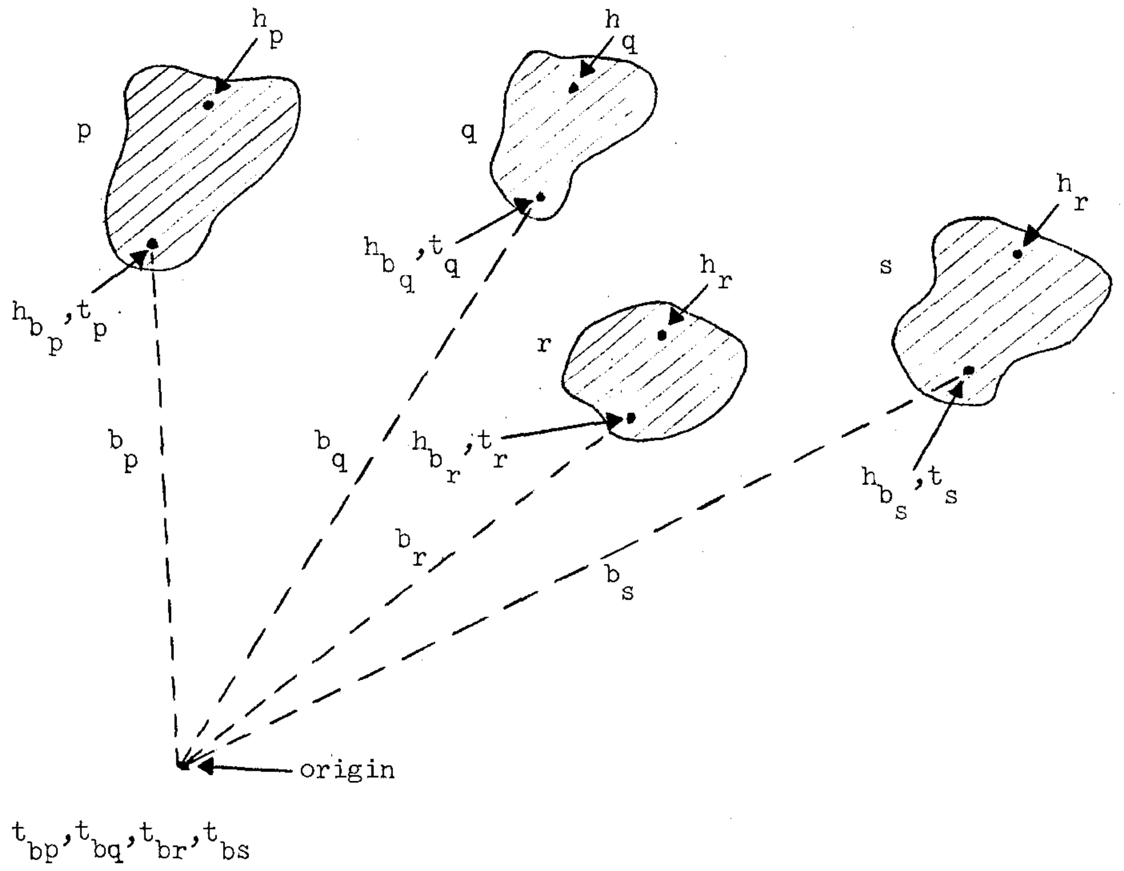


Figure 3.3(a) Labeled Picture

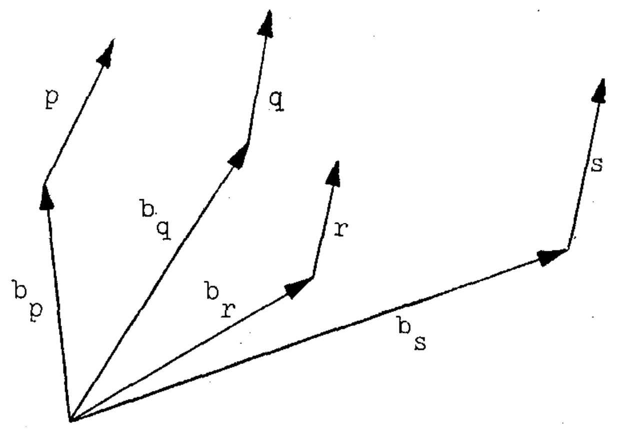


Figure 3.3(b) Corresponding Graph

Figure 3.3 An Extreme Case of a Connected Picture

3.3 PICTURE SYNTAX AND SEMANTICS

3.3.1 THE PDL LANGUAGE-PRIMITIVE DESCRIPTION OF A CONNECTED PICTURE

PDL (Picture Description Language) is a linear string language; a sentence S in PDL (expressed "$S \epsilon$PDL") provides the primitive structural description, T_s, of a picture by naming all its primitives (their class names) and their tail/head connectivity. The following syntax will generate any sentence $S \epsilon$PDL:

$$S \rightarrow p \mid S \phi_b S) \mid (\sim S) \mid SL \mid (/SL)$$
$$SL \rightarrow S^{\ell} \mid (SL \phi_b SL) \mid (\sim SL) \mid (/SL)$$
$$\phi_b \rightarrow + \mid \times \mid - \mid * \quad ,$$

where p may be any primitive class name (including λ) and ℓ is any label designator (represented by a lower case ALGOL ⟨identifier⟩). Any $S \epsilon$PDL will also be called a PDL expression.

Example:

$$\begin{aligned} T_s(\alpha) = (((\sim c) + ((a * ((b+a^i) + (\sim b))) + c)) \\ * ((a * ((b^i+a) + (\sim b^j))) \\ \times ((((/b^i) + (\sim c)) + ((/a^i) + c)) + (\sim(/b^j))))) \end{aligned}$$

for the picture α .

Not only primitives, but all pictures have a tail and a head; concatenations among pictures can occur only at their tail and head positions. Consider the picture α consisting of two subpictures α_1 and α_2 such that $\alpha_1 \epsilon \mathcal{P}(S_1)$, $\alpha_2 \epsilon \mathcal{P}(S_2)$ and $T_s(\alpha) = (S_1 \phi_b S_2)$, S_1, $S_2 \epsilon$ PDL . Then the tail and head of α according to $T_s(\alpha)$ is defined:

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

In the same way as primitives, more complex pictures are often represented by a directed edge of a graph. The interpretation of the binary concatenation operators is given in Table 3.1; in the table, the symbol cat means "is concatenated onto," and t and h indicate the tail and head of the resultant picture. The meaning of the concatenation operators may also be given by definitions of $\wp(T_s(\alpha))$; for example:

$$\wp((S_1+S_2)) = \{\alpha_1,\ \alpha_2 | \alpha_1 \in \wp(S_1) \wedge \alpha_2 \in \wp(S_2) \wedge \mathrm{head}(\alpha_1)\ \underline{\mathrm{cat}}\ \mathrm{tail}\ (\alpha_2)\}$$

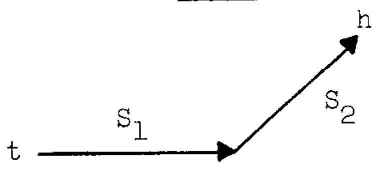
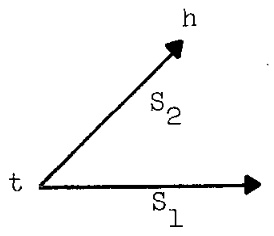
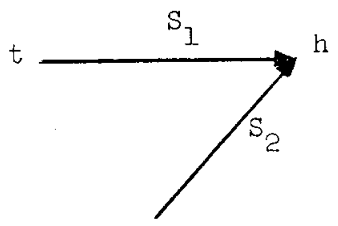
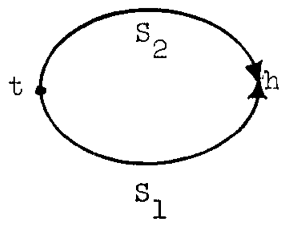
$T_s(\alpha)$	Interpretation	Graph
$(S_1 + S_2)$	$\mathrm{head}(\alpha_1)\ \underline{\mathrm{cat}}\ \mathrm{tail}(\alpha_2)$	
$(S_1 \times S_2)$	$\mathrm{tail}(\alpha_1)\ \underline{\mathrm{cat}}\ \mathrm{tail}(\alpha_2)$	
$(S_1 - S_2)$	$\mathrm{head}(\alpha_1)\ \underline{\mathrm{cat}}\ \mathrm{head}(\alpha_2)$	

Table 3.1 The Binary Concatenation Operators

Table 3.1 (continued)

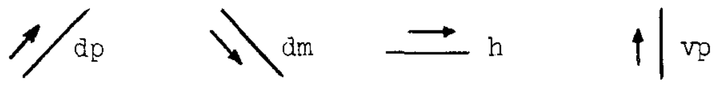
$T_s(\alpha)$	Interpretation	Graph
$(S_1 * S_2)$	$(\text{tail}(\alpha_1)\ \underline{\text{cat}}\ \text{tail}(\alpha_2))$ $\wedge\ (\text{head}(\alpha_1)\ \underline{\text{cat}}\ \text{head}(\alpha_2))$	

$$\alpha = \alpha_1 \cup \alpha_2$$
$$\alpha_1 \in P(S_1)$$
$$\alpha_2 \in P(S_2)$$

Figure 3.4 illustrates the use of these operators for describing a line drawing of an "A" and an "F"; typical members of each primitive class are shown with arrows pointing from the tail to the head positions. The structure of an "A" is built from its description.

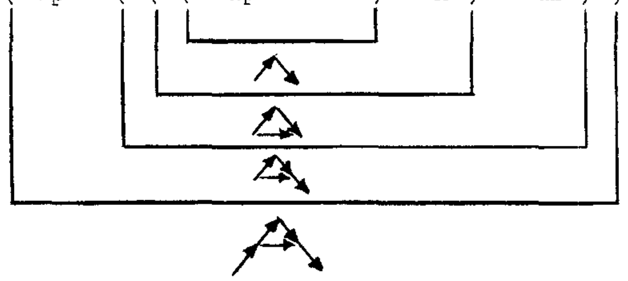
The connectivity graph of a PDL expression will often be referred to; in this case, the notation tail(S) and head(S) is used to indicate the tail and head nodes of the graph of the PDL expression S . Thus S and each picture in $P(S)$ has a tail and head position. tail(S) and head(S) will generically refer to both the pictures and the graph unless specifically stated otherwise.

Because of the freedom allowed in specifying primitive classes, a PDL expression may be undefined for some primitives. For example, if $P(\text{arc})$ is defined as in the example of section 3.2.2 and $P(\ell)$ is the



Primitive Classes

$$T_s(A) = (\,dp + (\,(\,(\,dp + dm\,) * h\,) + dm\,)\,)$$



$$T_s(F) = (vp + (h \times (vp + h)\,)\,)$$

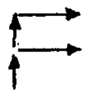


Figure 3.4 Primitive Structural Descriptions of an "A" and an "F"

class of all line segments with tail and head at their endpoints, then the concatenation expressed by $(\ell * \text{arc})$ can only have meaning for those members of $\wp(\ell)$ and $\wp(\text{arc})$ that are geometrically compatible; if $\wp(\text{arc})$ is restricted so that any chord is less than m units in length and $\wp(\ell)$ is restricted to lines of length greater than $2 \times m$, then $(\ell * \text{arc})$ is always undefined, i.e., $\wp((\ell * \text{arc}))$ is empty. This is no problem theoretically since the connectivity graph is constructed by treating each primitive abstractly, regardless of whether the concatenations are geometrically possible. It would, however, lead to undefined results in generation and failures in analysis of pictures.

This anomaly is ignored henceforth by allowing $\mathcal{P}(T_s)$ to define an empty set of pictures for some T_s .

The binary operators in conjunction with λ are sufficient to describe all possible tail/head concatenations between two pictures, i.e., they are locally complete; Figure 3.5 enumerates and describes all possible local concatenations. The unary operators $\sim$ and $/$ do not describe concatenations, but allow the tail and head to be moved. A notation of description equivalence is introduced in order to discuss the unary operators, labeled expressions, and some formal properties of PDL. For $S_1, S_2 \in$ PDL:

1. S_1 is <u>weakly equivalent</u> to S_2 $(S_1 \equiv_w S_2)$ if there exists an isomorphism between the graphs of S_1 and S_2 such that corresponding edges have identical names.
2. S_1 is equivalent to S_2 $(S_1 \equiv S_2)$ if
   a. $S_1 \equiv_w S_2$, and
   b. $\text{tail}(S_1) = \text{tail}(S_2)$
   and $\text{head}(S_1) = \text{head}(S_2)$.

The unary $\sim$ operator acts as a tail/head reverser with the following properties:

1. $(\sim S_1) \equiv_w S_1, \qquad S_1 \in$ PDL
2. $\text{tail}((\sim S_1)) = \text{head}(S_1)$ and $\text{head}((\sim S_1)) = \text{tail}(S_1)$.

The purpose of PDL expressions with label designators, such as S^ℓ, is to allow cross-reference to that expression within a description; with the $/$ operator, this enables the tail and head to be arbitrarily located. A PDL expression S^ℓ is equivalent to the value of the following function g:

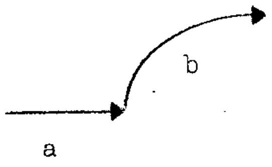
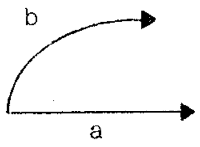
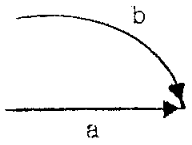
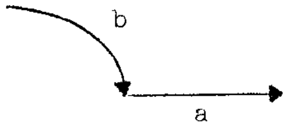
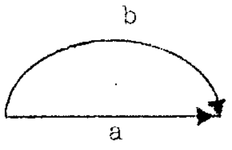
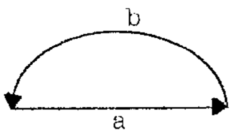
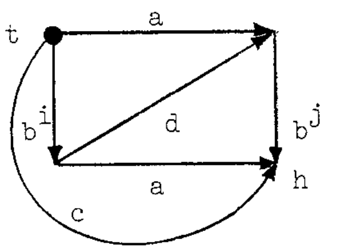
Concatenation	Description
	(a + b)
	(a X b)
	(a − b)
	(b + a)
	(a * b)
	((a+b) * λ)

Figure 3.5 Local Completeness of {+, X, -, *}

```
g(S^ℓ) = if primitive(S) then S^ℓ
         else
         if S = (S_1 ∅_b S_2), ∅_b ∈ {+, ×, -, *},
                 then (g(S_1^ℓ) ∅_b g (S_2^ℓ))
         else
         if S = (∅_u S_1), ∅_u ∈ {~, /},
                 then (∅_u g(S_1^ℓ)),
```

where primitive(S) = true if (1) S is a primitive class name, or (2) $S = S_1^{\ell}$ where S_1 is a primitive class name, and false otherwise. Concatenated label designators are interpreted as single labels; thus $((a^{i}+b)^{j} + a^{i}) \equiv ((a^{ij}+b^{j}) + a^{i})$.

Figure 3.6 illustrates the use of label designators and the / operator to describe (a) a picture whose connectivity is equivalent to that of the complete 4-node graph, and (b) a line drawing of a three-dimensional cube in 3-space; in the latter, the primitives are line segments in the X, Y, and Z directions, where the Z direction points into the paper. The explanation of the / operator assumes that any expression S^{ℓ} within a PDL expression has been recursively transformed by the above function g into an equivalent expression so that only primitives have label designators. Then it is required that each primitive within the scope of a / operator, i.e., each primitive that is part of some (/S) within the PDL expression, have a label designator (this is part of the PDL syntax given earlier) and be identical in name and label to one and only one primitive outside the scope of a / . The / is interpreted as a superposition or blanking operator. Each primitive within its scope is another instance of its identical



$$T_s(\alpha) = (((b^i + a) * (((/b^i) + d) + (/b^j))) * ((a + b^j) * c))$$

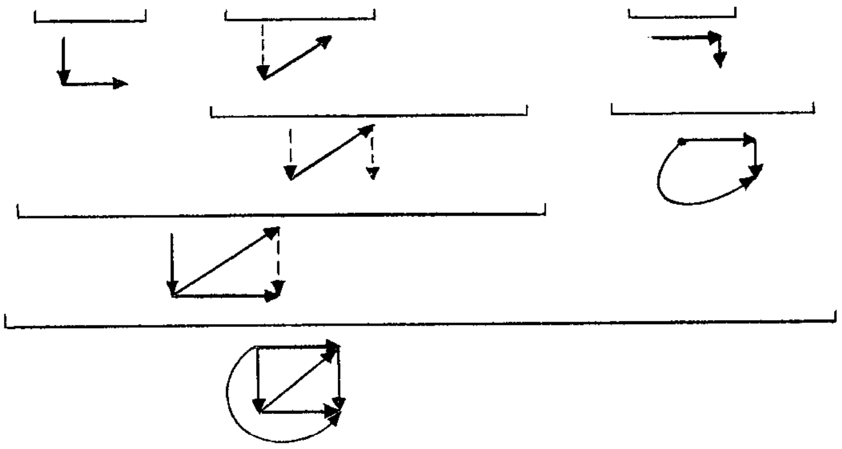
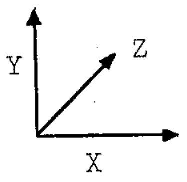
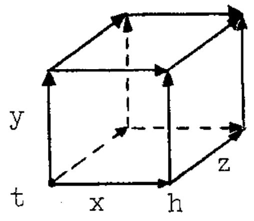


Figure 3.6(a) The Complete 4-Node Graph with Directed Edges





$$\begin{aligned} T_s(\alpha) = {} & (((x * ((y^i + x) + (\sim y^j))) \\ & * (((((/y^i) + z) + ((x * ((\sim y) + (x^k + y))) \\ & + (\sim z))) + (\sim (/y^j)))) \\ & * ((z + (/x^k)) + (\sim z))) \end{aligned}$$

Figure 3.6(b) A 3-Dimensional Cube

Figure 3.6 PDL Descriptions with Labels and /

outside primitive; the description of concatenations onto either one will refer to the <u>same</u> primitive. Thus $/$ allows multiple descriptions of the same primitives and structures, effectively moving the tail or head to a more convenient place for further concatenations. A formal definition of the meaning of $/$ and label designators is given in section 3.4.2.

It is now possible to state completely the rules for determining the tail and head of an expression $S \in \text{PDL}$ and of each $\alpha \in \mathcal{P}(S)$:

$$
\begin{array}{l}
\begin{Bmatrix}\text{tail}(S)\\ \text{head}(S)\end{Bmatrix} = \underline{\text{if}}\ \ \text{primitive}(S)\ \ \underline{\text{then}}\ \begin{Bmatrix}\text{tail}(S)\\ \text{head}(S)\end{Bmatrix}\\
\qquad\qquad\quad \underline{\text{else}}\\
\qquad\qquad\quad \underline{\text{if}}\ \ S = (S_1 \emptyset_b S_2),\ \ \emptyset_b \in \{+, \times, -, *\}\\
\qquad\qquad\qquad\qquad \underline{\text{then}}\ \begin{Bmatrix}\text{tail}(S_1)\\ \text{head}(S_2)\end{Bmatrix}\\
\qquad\qquad\quad \underline{\text{else}}\\
\qquad\qquad\quad \underline{\text{if}}\ \ S = (\sim S_1)\ \ \underline{\text{then}}\ \begin{Bmatrix}\text{head}(S_1)\\ \text{tail}(S_1)\end{Bmatrix}\\
\qquad\qquad\quad \underline{\text{else}}\\
\qquad\qquad\quad \underline{\text{if}}\ \ S = S_1^{\ell}\ \ \underline{\text{then}}\ \begin{Bmatrix}\text{tail}(g(S_1^{\ell}))\\ \text{head}(g(S_1^{\ell}))\end{Bmatrix}\\
\qquad\qquad\quad \underline{\text{else}}\\
\qquad\qquad\quad \underline{\text{if}}\ \ S = (/S_1)\ \ \underline{\text{then}}\ \begin{Bmatrix}\text{tail}(S_1)\\ \text{head}(S_1)\end{Bmatrix}
\end{array}
$$

where the function g is defined earlier.

The <u>primitive semantic</u> or <u>value description</u> $T_v(\alpha)$ of a picture α is a list of the descriptions $D(\beta)$ of those primitive pictures β contained in α which have their connectivity and class names described in $T_s(\alpha)$.

<u>Example</u>: A picture α consisting of a straight line segment concatenated onto an endpoint of an arc might have:

$$T_s(\alpha) = (\text{line} + \text{arc})$$

$$T_v(\alpha) = ((\text{line}, ((x_1, y_1), (x_2, y_2), m)),$$
$$(\text{arc}, ((x_2, y_2), (x_3, y_3), r))),$$

where m is the line slope and r is the radius of the arc-generating circle.

One more assumption is necessary in order to complete the PDL description scheme. It is assumed that all pictures have a well-defined origin from which a PDL description "starts"; that is, any PDL description S of a picture is interpreted as $(\lambda + S)$ where the tail and head of λ is the picture origin. The origin can be any convenient point in the picture and is usually determined by either the digitization or the generation mechanism. In analysis problems, this normally means that the first primitive concatenated onto the origin is a blank or "don't care" primitive whose recognition function is equivalent to a search strategy to find some interesting visible part of the picture.

3.3.2 HIERARCHIC DESCRIPTIONS

The set of rules or grammar $\mathcal{G}$ that describes (generates) the class of pictures $P_{\mathcal{G}}$ will be a type 2 (context-free) phrase structure grammar (Chomsky [1959]) with the following restrictions. Each rule or production is of the form:

$$S \rightarrow pdl_1 | pdl_2 | \text{-----} | pdl_n, \qquad n \geq 1,$$

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. Sentences of $\mathcal{L}(\mathcal{G})$ will consist of PDL expressions; thus, the class of terminal symbols of $\mathcal{G}$ will be a subset of

$$\{+, \times, -, *, \sim, /, (,)\} \cup \left\{\begin{array}{c}\text{primitive}\\ \text{class}\\ \text{names}\end{array}\right\} \cup \left\{\begin{array}{c}\text{label}\\ \text{designators}\end{array}\right\}.$$

Non-terminal symbols are denoted by upper case ALGOL ⟨identifiers⟩s . Each grammar $\mathcal{G}$ will have one <u>distinguished</u> non-terminal symbol from which $\mathcal{L}(\mathcal{G})$ 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.

The <u>hierarchic</u> <u>structural</u> description $H_s(\alpha)$ of a picture $\alpha \in P_{\mathcal{G}}$ having structural description $T_s(\alpha) \in \mathcal{L}(\mathcal{G})$ is defined as the parse of $T_s(\alpha)$ according to $\mathcal{G}$; $H_s(\alpha)$ is conveniently represented as a parenthesis-free tree. A simple example is given in Figure 3.7.

$$\begin{array}{lrcl}
\mathcal{G}: & P & \rightarrow & A \mid HOUSE \\
 & A & \rightarrow & (dp + (TRIANGLE + dm)) \\
 & HOUSE & \rightarrow & ((vm + (h + (\sim vm))) * TRIANGLE) \\
 & TRIANGLE & \rightarrow & ((dp + dm) * h)
\end{array}$$

$$\begin{array}{rl}
\mathcal{L}(\mathcal{G}) = & \{(dp + (((dp + dm) * h) + dm)), \\
 & ((vm + (h + (\sim vm))) * ((dp + dm * h))\}
\end{array}$$

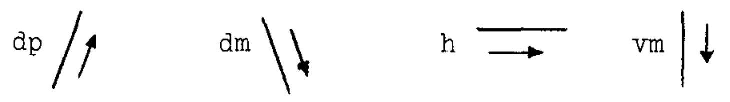
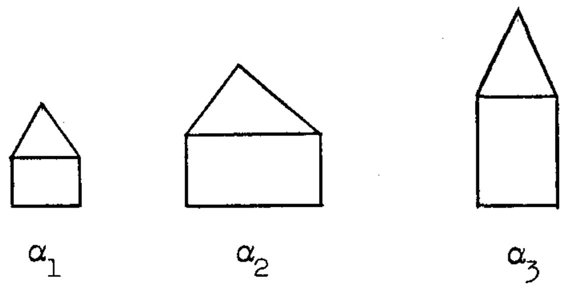


Figure 3.7(a) $\mathcal{G}$, $\mathcal{L}(\mathcal{G})$, and Primitives



$T_s(\alpha_i) = ((vm + (h + (\sim vm))) * ((dp + dm) * h))$

$H_s(\alpha_i)$:

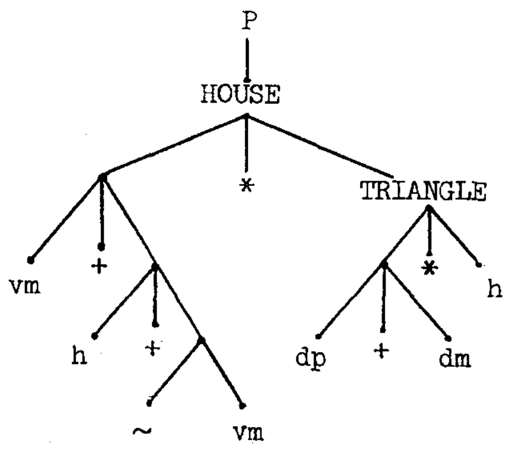


Figure 3.7(b) Examples and Parse of a "P"

Figure 3.7 Structural Descriptions of a Picture

The use of a formal grammar to describe picture classes has several advantages. Alternatives in a production allow the same name to be assigned to different structures that belong to the same pattern class. Large classes of similarly structured pictures can be concisely defined by recursive productions. For example, all tree structures with "branches" from primitive class b can be defined by the syntax:

$$\text{TREE} \rightarrow b \,|\, (b + \text{TREE}) \,|\, (\text{TREE} \times \text{TREE})$$

Nodes or points in a picture may be named (and assigned properties by $\mathcal{I}$) by rules of the form:

$$\text{NODE} \rightarrow \lambda$$

The rationale behind the selection of context-free grammars rather than more complex ones is mainly one of simplicity; their form is simple, they can generate PDL descriptions for a large, useful, and interesting class of pictures, and there is a great deal of theoretical and practical knowledge on their use in the description and analysis of string languages (Ginsburg [1966], Feldman and Gries [1967]).

Corresponding to each rule of $\mathcal{S}$ will be a <u>semantic</u> rule in $\mathcal{I}$. Two sets of semantic rules are postulated--a <u>natural</u> semantics $\mathcal{I}_n$ and an <u>imposed</u> semantics $\mathcal{I}_m$. The natural semantics $H_v(\alpha)$ of a picture α is a list containing the name, tail, and head of each non-terminal symbol (syntax rule) in $H_s(\alpha)$, where the tail and head of a non-terminal symbol is defined as the tail and head of the PDL expression generated by it. Any α_i, $i = 1, 2, 3$, in Figure 3.7 would have:

$$H_v(\alpha_i) = ((P, (t, h)_P), (\text{HOUSE}, (t, h)_{\text{HOUSE}}), (\text{TRIANGLE}, (t, h)_{\text{TRIANGLE}})),$$

where $(t, h)_k$ is the tail and head of k.

The purpose of an imposed semantics is to take an action and assign a value or set of values to a non-terminal symbol upon successful application of its syntax rule during a parse. This action might be to compute a function over the structures or picture described by the syntax rule or to generate code for later execution. The last case would occur when $\mathcal{S}$ and $\mathcal{I}_m$ were input to a compiler/compiler (Feldman [1966], Reynolds [1965]); the output would then be a picture processing system for $\mathcal{P}_{\mathcal{G}}$. A mechanism to express elements of $\mathcal{I}_m$ has not been developed; the natural semantics only is used here.

The description scheme for pictures can now be summarized:

The class of pictures of interest is generated by a given grammar $\mathcal{G}$ such that

$$\mathcal{P}_{\mathcal{G}} = \bigcup_{T_s \in \mathcal{L}(\mathcal{G})} \mathcal{P}(T_s) ,$$

and

$$\mathcal{L}(\mathcal{G}) \subseteq \text{PDL} .$$

Then, the description of $D(\alpha)$ of any picture $\alpha \in \mathcal{P}_{\mathcal{G}}$ is

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

where $T_s(\alpha) \in \mathcal{L}(\mathcal{G})$,

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

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

$H_v(\alpha)$ is the natural semantics of α .

3.4 PDL: FORMAL PROPERTIES AND BASIC THEOREMS

3.4.1 ALGEBRAIC PROPERTIES

The definition and interpretation of the PDL language can be viewed as a picture or graph algebra over the set of primitive structural descriptions under the operations $+, -, \times, *, \sim,$ and $/$. Elements (sentences $S \in \text{PDL}$) are considered equal if they are equivalent. A number of useful algebraic properties are given below; it is assumed that

$S, S_1, S_2, S_3 \in \text{PDL}, \emptyset_b \in \{+, \times, -, *\}, \emptyset_{b-*} \in \{+, \times, -\}$.

1. Associativity:

Each of the binary concatenation operators is associative.

(a) $((S_1 + S_2) + S_3) \equiv (S_1 + (S_2 + S_3))$

(b) $((S_1 \times S_2) \times S_3) \equiv (S_1 \times (S_2 \times S_3))$

(c) $((S_1 - S_2) - S_3) \equiv (S_1 - (S_2 - S_3))$

(d) $((S_1 * S_2) * S_3) \equiv (S_1 * (S_2 * S_3))$

This allows the elimination of parentheses from an expression whose operators are identical. Thus, $((S_1 + S_2) + S_3)$ can be put in the simpler form $(S_1 + S_2 + S_3)$, and $(S_1 - S_2) - S_3)$ in the form $(S_1 - S_2 - S_3)$.

2. Commutativity:

(a) $*$ is the only commutative binary operator.

$$(S_1 * S_2) \equiv (S_2 * S_1)$$

(b) $\times$ and $-$ are "weakly" commutative.

$$(S_1 \times S_2) \equiv_w (S_2 \times S_1)$$

$$(S_1 - S_2) \equiv_w (S_2 - S_1)$$

3. The $\sim$ Operator:

$\sim$ acts much like complementation in a Boolean algebra.

(a)

$$(\sim(S_1 + S_2)) \equiv ((\sim S_2) + (\sim S_1))$$

$$(\sim(S_1 * S_2)) \equiv ((\sim S_2) * (\sim S_1))$$

(b) $\sim$ obeys a "de Morgan's law" with respect to $\times$ and $-$:

$$(\sim(S_1 \times S_2)) \equiv ((\sim S_2) - (\sim S_1))$$

$$(\sim(S_1 - S_2)) \equiv ((\sim S_2) \times (\sim S_1))$$

Note that $\sim$ reverses the order of the operands. The equivalences of (a) and (b) are useful for moving the $\sim$ within an expression.

(c) Involution:

$$(\sim(\sim S)) \equiv S$$

4. <u>The / Operator</u>:

(a) $(/(/S)) \equiv (/S)$

(b) $(/(S_1 \phi_b S_2)) \equiv ((/S_1) \phi_b (/S_2))$

(c) $(/(\sim S)) \equiv (\sim(/S))$

5. <u>The Null Point Primitive λ</u>:

(a) $(S\phi_b\lambda) \equiv (\lambda\phi_b S)$

(b) $(S\phi_{b-*}\lambda) \equiv S$

$(S * \lambda) \not\equiv S$ since $(S * \lambda)$ implies $\text{head}(S) = \text{tail}(S)$

(c) $(\sim \lambda) \equiv \lambda$

(d) $(\lambda \phi_b \lambda) \equiv \lambda$

3.4.2 THE GRAPH OF A PDL EXPRESSION

By using some of the algebraic properties of the last section to move unary operators and label designators as far as possible within an expression, a <u>standard form</u> $f(S) \in PDL$ of an expression S can be obtained. $f(S)$ is defined:

```
f(S) = if  $(S = S_1 \vee S = (/S_1) \vee S = (\sim S_1) \vee S = (\sim(/S_1))) \wedge \text{primitive}(S_1)$
           then  $S$
       else
       if  $S = (S_1 \phi_b S_2)$, $\phi_b \in \{+, \times, -, *\}$,  then  $(f(S_1)\ \phi_b\ f(S_2))$
       else
       if  $S = S_1^{\ell}$  then  $f(g(S))$
       else
       if  $S = (\sim(S_1 \phi S_2))$, $\phi \in \{+, *\}$,  then  $(f((\sim S_2))\ \phi\ f((\sim S_1)))$
       else
       if  $S = (\sim(S_1 \times S_2))$  then  $(f((\sim S_2)) - f((\sim S_1)))$
       else
       if  $S = (\sim(S_1 - S_2))$  then  $(f((\sim S_2)) \times f((\sim S_1)))$
       else
       if  $S = (/(S_1 \phi_b S_2))$, $\phi_b \in \{+, \times, -, *\}$,  then
               $(f((/S_1))\ \phi_b\ f((/S_2)))$
       else
       if  $S = (\sim(\sim S_1))$  then  $f(S_1)$
       else
       if  $S = (\sim(/S_1)) \vee S = (/(\sim S_1))$  then  $f((\sim f((/S_1))))$
       else
       if  $S = (/(/S_1))$  then  $f((/S_1))$
```

Example:

$$((\sim(a^i + b)) + (/(\sim a^i)))$$

has the standard form:

$$(((\sim b) + (\sim a^i)) + (\sim(/a^i)))$$

The standard form $f(S)$ of S has the properties:

1. $f(S) \equiv S$,
2. the operand of each $/$ is a primitive class name, and
3. the operand of each $\sim$ is either a primitive class name or $/$ followed by a primitive class name.

The function definition is a case analysis of all possible forms of S as given by the PDL syntax.

A <u>valid</u> PDL expression (vPDL) is one whose standard form is such that if $(/p^{\ell})$ appears in it one or more times for some primitive p and label ℓ, then p^{ℓ} also appears once and only once outside the scope of a $/$.

The graph, and therefore the primitive connectivity, described by a vPDL S is defined by the following algorithm:

1. Transform S into standard form by applying the function f .
2. Replace each expression of the form $(/p^{\ell})$ by a new primitive $p^{\ell}_{/}$. This removes all $/$ operators.
3. Generate the connectivity graph of the resulting expression.
4. Contract the tail and head nodes of each edge $p^{\ell}_{/}$ to the corresponding nodes of p^{ℓ} .
5. Eliminate all edges of the form $p^{\ell}_{/}$.

The above algorithm formally defines the meaning of labeled expressions and the $/$ operator. A simple example is given in Figure 3.8.

$((((a^{i} + b) * (b + a)) * c) + (/a^{i}))$

$$\xRightarrow{\text{step 2}} \quad ((((a^{i} + b) * (b + a)) * c) + a^{i}_{/})$$

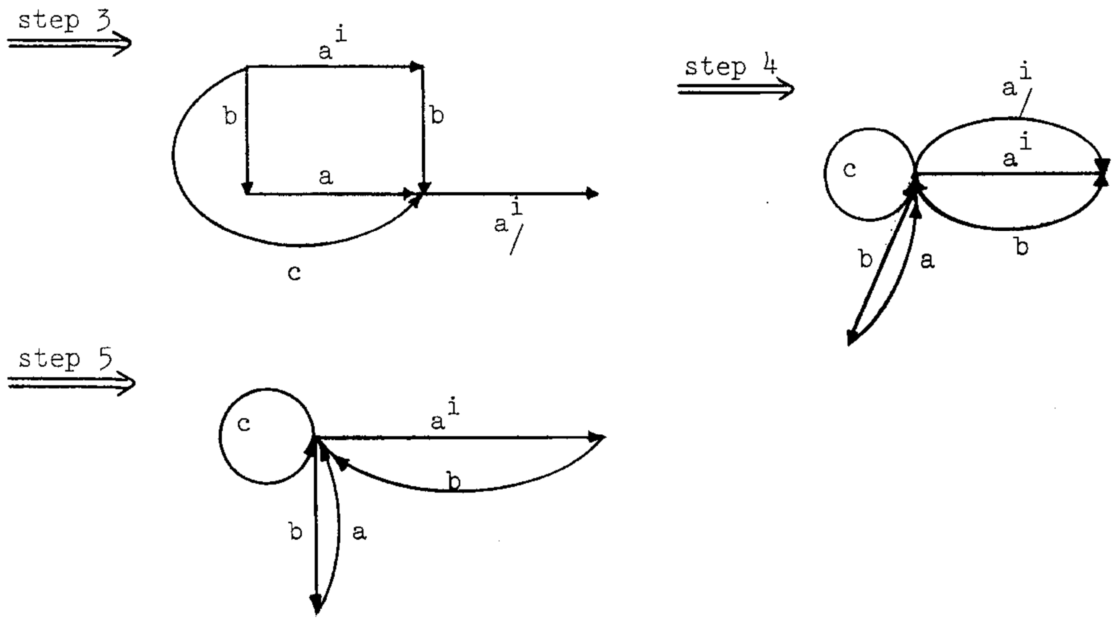


Figure 3.8 The Graph of a vPDL

3.4.3 BASIC THEOREMS

1. <u>Connectivity Description</u>

Each step in the formation of a graph of a vPDL can always be performed and has a unique result. This leads to:

<u>THEOREM 3.1</u>:

Any vPDL describes a unique primitive connectivity.

This gives the assurance that one and only one primitive connectivity is represented by a vPDL .

2. Completeness

THEOREM 3.2:

Any connected set of primitives can be effectively described by a vPDL .

Proof:

The proof is by induction on the number n of connected primitives. For $n = 1$, the vPDL is p, where p is the primitive class name. Suppose that any connected set of n primitives can be effectively described by a vPDL .

Consider $(n + 1)$ connected primitives. Select n of these that are connected, say $p_1, p_2, \ldots, p_n$. By the induction hypothesis, their connectivity may be described by a vPDL:

$$S_n = S_n(p_1, p_2, \ldots, p_n)$$

(a) The first possibility is that the $(n + 1)$st primitive, p_{n+1}, is connected by only one of its nodes to a primitive in S_n . Then, there must exist at least one p_i, $1 \leq i \leq n$, whose tail or head, or both are connected to p_{n+1} .

The following connectivities are possible:

(1) head(p_i) to head(p_{n+1})
(2) tail(p_i) to head(p_{n+1})
(3) head(p_i) to tail(p_{n+1})
(4) tail(p_i) to tail(p_{n+1})

Consider case (1):

Since $p_1, p_2, \ldots, p_n$ are connected, a "path", described by S_i, can be found from head(S_n) to head(p_i), such that:

tail(S_i) = head(S_n) and head(S_i) = head(p_i) . (Figure 3.9(a).)

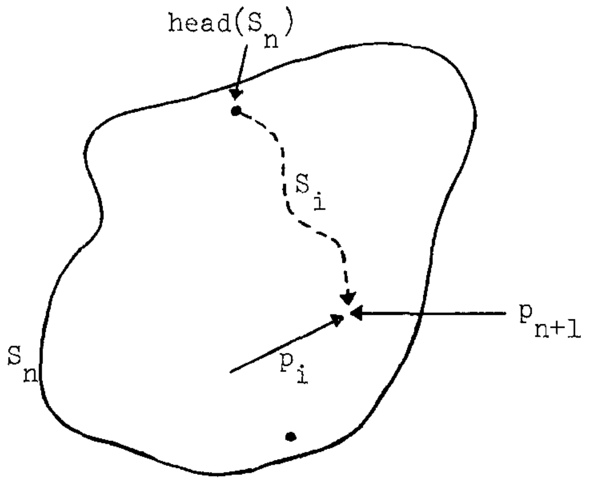


Figure 3.9(a) head(p_{n+1}) <u>cat</u> head(p_i)

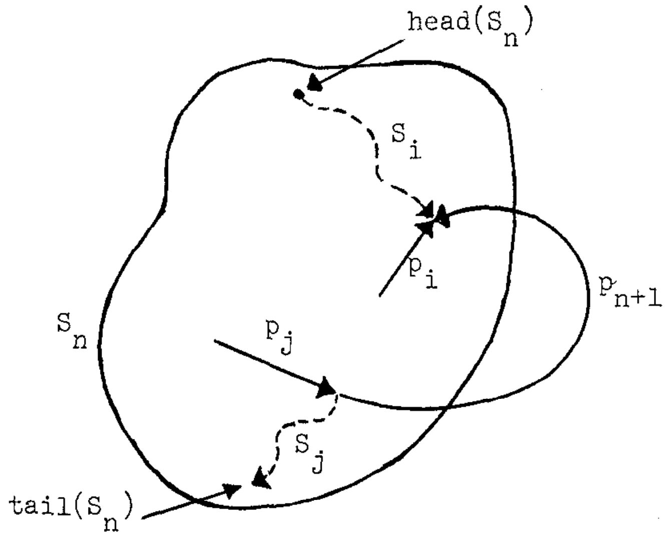


Figure 3.9(b) head(p_{n+1}) <u>cat</u> head(p_i) $\wedge$ tail(p_{n+1}) <u>cat</u> head(p_j)

Figure 3.9 Theorem 3.2

The form of S_i can be:

$$S_i = (S_{i1} + S_{i2} + \ldots + S_{in_i}),$$

where $S_{ij} = (\sim p_{ij})$ or $S_{ij} = p_{ij}$ and $p_{ij} \in \{p_i, p_2, \ldots, p_n\}$ $j = 1, 2, \ldots, n_i$.

(Parentheses are omitted in S_i since $+$ is associative.)

All the primitives in S_i which are labeled in S_n are now given the same label in S_i . Call the resulting expression S_i^1 .

Label uniquely all the unlabeled primitives of S_i^1 ; attach the same labels to the corresponding primitives in S_n . Call the resulting expressions SL_i and SL_n respectively. Then the following vPDL describes the connectivity of the $(n + 1)$ primitives:

$$S_{n+1} = ((SL_n + (/SL_i)) - p_{n+1})$$

The remaining cases are handled by a similar construction.

(b) The only other possibility is that p_{n+1} is connected at both of its nodes to S_n . Therefore, there exist p_i and p_j, $1 \leq i, \ j \leq n$, such that:

(1) $\text{head}(p_i) = \text{head}(p_{n+1}) \wedge (\text{head}(p_j) = \text{tail}(p_{n+1})$
$\qquad\qquad\qquad\qquad \vee \ \text{tail}(p_j) = \text{head}(p_{n+1}))$

or

(2) $\text{head}(p_i) = \text{tail}(p_{n+1}) \wedge (\text{head}(p_j) = \text{head}(p_{n+1})$
$\qquad\qquad\qquad\qquad \vee \ \text{tail}(p_j) = \text{head}(p_{n+1}))$

or

$$(3)\quad \text{tail}(p_i) = \text{head}(p_{n+1}) \wedge (\text{head}(p_j) = \text{tail}(p_{n+1}) \vee \text{tail}(p_j) = \text{tail}(p_{n+1}))$$

or

$$(4)\quad \text{tail}(p_i) = \text{tail}(p_{n+1}) \wedge (\text{head}(p_j) = \text{head}(p_{n+1}) \vee \text{tail}(p_j) = \text{head}(p_{n+1}))$$

Consider the case:

$$\text{head}(p_i) = \text{head}(p_{n+1}) \wedge \text{head}(p_j) = \text{tail}(p_{n+1})$$

As in (a), there is a path, described by S_i, from head(S_n) to head(p_i); similarly, there is a path that can be described by S_j from head(p_j) to tail(S_n) . S_i and S_j satisfy:

$$\text{head}(S_i) = \text{head}(p_i),\ \text{tail}(S_i) = \text{head}(S_n)$$
$$\text{head}(S_j) = \text{tail}(S_n),\ \text{tail}(S_j) = \text{head}(p_j)\ .$$

(See Figure 3.9(b), page 52.) The same labeling as in (a) is done except that any primitive common to S_i, S_j, and S_n is labeled in all three expressions. Call the resulting expressions SL_i, SL_j, and SL_n . Then the connectivity of the $(n + 1)$ primitives is described by the vPDL:

$$S_{n+1} = (((/SL_j) + (SL_n + (/SL_i))) * p_{n+1})\ .$$

The other cases are treated in a similar manner. Therefore, the case of $(n + 1)$ connected primitives is proven.

Q.E.D.

Note that, in general, more than one vPDL can be obtained to describe the same connectivity; for example, in part (b) of the proof, a similar argument would yield the vPDL:

$$S'_{n+1} = ((SL_n + (/SL_i)) * ((\sim(/SL_j)) + p_{n+1})) \ .$$

Corollary 3.1 (Linear Cipher):

Any directed graph can be described by a vPDL .

Theorem 3.1 proves the completeness of a PDL with respect to the primitive structural description of any connected set of primitives. Corollary 3.1 further suggests that graphs of various types may be represented and possibly manipulated within PDL.

3. Moving the Tail and Head

The path construction used in the proof of Theorem 3.2 can be employed to move the tail and/or head of a vPDL to any node(s) in the structure.

THEOREM 3.3:

Given a vPDL S_1 describing a set of connected primitives whose corresponding graph has n nodes, it is possible to derive n^2-1 (and no more) other vPDL's, $S_2, S_3, \ldots, S_{n^2}$, such that

$$\left.\begin{array}{ll}(1) & S_i \equiv_w S_j \\ & \\ (2) & S_i \not\equiv S_j\end{array}\right\} \quad \begin{array}{l} i,\ j = 1, 2, \ldots, n^2 \\ i \neq j \end{array}$$

(3) Each S_i, $i = 2, \ldots, n^2$, is equivalent to an expression having one of the forms:

(a) $((/S_{i1}) + (SL_1 + (/S_{i2})))$

(b) $((/S_{i1}) + SL_i)$

(c) $((SL_1 + (/S_{i2}))$,

where SL_1 is obtained from S_1 by giving the same labels to those primitives in S_1 that appear in S_{i1} and/or S_{i2}.

Proof:

Since there are n nodes in the graph, there are n^2 different ways of assigning the tail and head. Therefore, given S_1 with its tail and head, there are n^2-1 other assignments that can be made. Since the primitives are connected, a path can always be found from the desired tail to tail(S_1) and from head(S_1) to the desired head. Using the construction in the proof of Theorem 3.2, expressions of the form (a) (or (b) when the new head = head(S_1), or (c) when the new tail = tail(S_1)), can always be derived. Properties (1) and (2) follow immediately.

Theorem 3 allows one to take the origin (tail) of a picture at any convenient place. It also assures access to any node in the graph when building up descriptions.

4. An Adequate and Independent Set of Operators

The question naturally arises whether label designators and the / operator are necessary or just convenient. Theorem 3.4 proves the inadequacy of the system without these features.

THEOREM 3.4:

The operator set $\{+, \times, *, -, \sim\}$ is not sufficient for the description of any connected set of primitives.

Proof:

Assume that / is not part of the PDL language. If $(S_1\{\overset{+}{\underset{-}{\times}}\}S_2)$ is contained in a vPDL S, the nodes

$$\left\{\begin{array}{c}\text{head}(S_1)(=\text{tail}(S_2))\\ \text{head}(S_1)\\ \text{tail}(S_2)\end{array}\right\}$$

are inaccessible within S since only $\text{tail}(S_1)$ and $\text{head}(S_2)$ can be used for further concatenations in S (by definition); furthermore, the inaccessible node has at most two edges meeting at it. Consider a picture whose connectivity is equivalent to that of the complete 4 node graph (Figure 3.6(a)); let each edge have the name x . Then any description S of this connectivity must contain a subexpression equivalent to $(X_1\phi_{b-*}X_2)$, where $X_1 = (\sim x)$ or x, $X_2 = (\sim x)$ or x, and $\phi_{b-*} \in \{+, \times, -\}$. $*$ is not possible since $(X_1 * X_2)$ does not describe any subgraph of the graph; this also applies to $(X_1 * \lambda)$. Finally, if only expressions of the form $(X_1\phi_{b-*}\lambda)$ appeared, the equivalence $(X_1\phi_{b-*}\lambda) \equiv X_1$ could be applied to obtain the above form. But, each node must have 3 edges meeting at it. Since the expression $(X_1\phi_{b-*}X_2)$ leaves one node inaccessible with at most two edges tied onto it, S cannot describe the picture.

Q.E.D.

However, there does exist an adequate and independent set of operators.

THEOREM 3.5:

Any vPDL is equivalent to one that uses only the operator set $\{+, \sim, /\}$. Moreover, these operators are independent.

Proof:

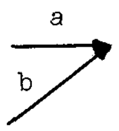
The following equivalent expressions demonstrate the adequacy of $\{+, \sim. /\}$:

$$(S_1 * S_2) \equiv ((S_1^i + (\sim S_2)) + (/S_1^i))$$

$$(S_1 \times S_2) \equiv ((S_1^i + (/(\sim S_1^i))) + S_2)$$

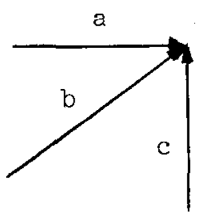
$$(S_1 - S_2) \equiv ((S_1 + (\sim S_2^i)) + (/S_2^i)),$$

where i does not appear as a label in S_1 or S_2. $+$ is independent of $\sim$ and $/$, since it is the only concatenation operator. $\sim$ is independent since $+$ and $/$ cannot be used to describe the connectivity:



$(a + (\sim b))$

$/$ is independent since $\sim$ and $+$ cannot alone describe the connectivity:



$((a + (\sim b^i)) + ((/b^i) + (\sim c)))$

Q.E.D.

The set $\{\times, -, *\}$, while unnecessary, is still very convenient, especially in the description of pictures with simple structure.

CHAPTER 4

THE FORMAL DESCRIPTION OF SEVERAL PICTURE CLASSES

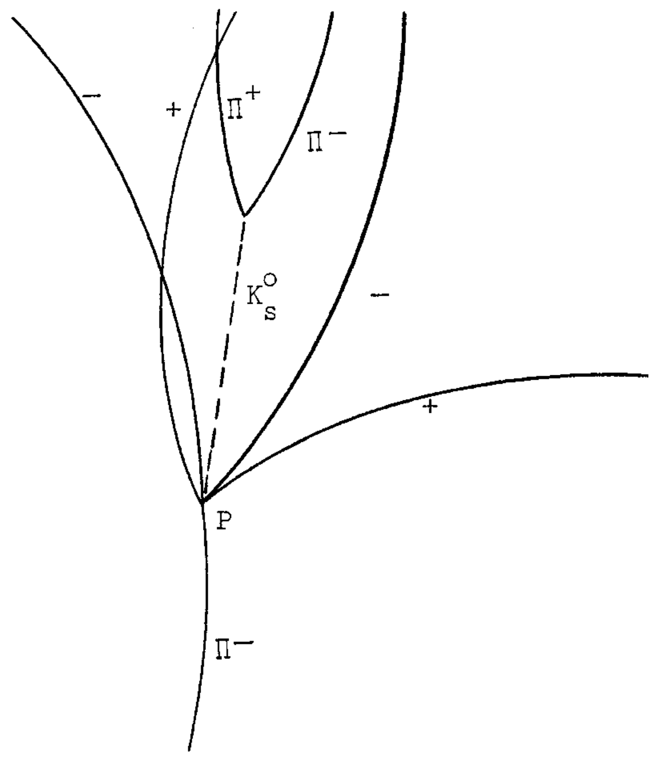
The examples of this chapter illustrate both the power and the limitations of the PDL system as a formal picture description scheme. Comparisons with some of the work surveyed in section 2.3 are made where appropriate.

Primitive classes are defined informally by a pictorial sample, a mnemonic name, and often a textual description, rather than by a detailed definition of their recognition (or generation) functions. The latter depends to a great extent on factors that are irrelevant at this point; these include the amount of noise in a particular picture, the hardware used for reading and displaying pictures, and the eventual purpose of the description.

4.1 PARTICLE PHYSICS

In high energy particle physics, one of the most common methods for obtaining the characteristics of an elementary particle is to analyze the trajectory "trail" left by the particle and its byproducts in a detector chamber, such as a bubble or spark chamber (Shutt [1967]). Several hundred thousand photographs of these trails might be taken in a typical experiment. Because of the large numbers involved and the accuracy and quantity of computation required for each "interesting" photograph, machine processing of the pictures is desirable.

Figure 4.1 contains a photograph of a typical event occurring in a bubble chamber. In the left central part of the picture, the following track configuration can be seen:



A Π^- particle entering from the bottom interacts with a proton P producing two positive particles (+), two negative particles (-), and an unseen neutral particle (K°_s) . The neutral particle later decays and produces the pair Π^- and Π^+ .

In addition to the particle tracks, the pictures usually contain some identifying information (in a "data box"), such as frame number, view, input beam characteristics, and date, and a set of "fiducials", which are marks on the chamber whose positions are precisely known. Fiducials allow the tracks to be reconstructed in real space.

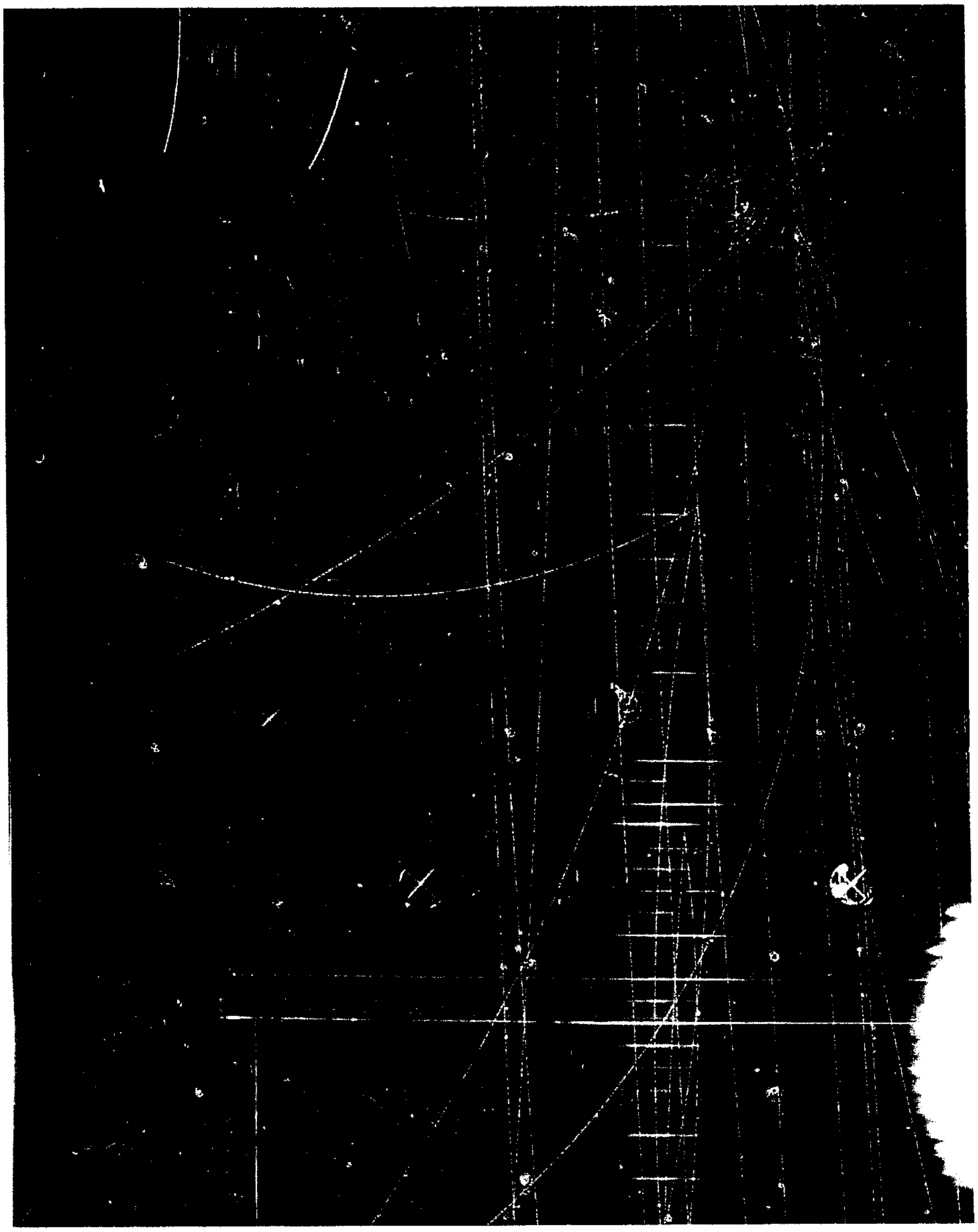


Figure 4.1 Event in Bubble Chamber

Photo By Courtesy of Dr. William Johnson, SLAC

Figure 4.2 gives the syntax for an abstracted particle physics picture. A negatively charged particle TM is assumed to enter a chamber containing positive particles P and under the influence of a magnetic field; TM enters from the left. The following types of reactions are provided for:

(a) Interaction with P:

$$
\begin{aligned}
TM + P &\rightarrow TM + TP \\
&\rightarrow TM + TP + TN \\
&\rightarrow TN
\end{aligned}
$$

(b) Negative Particle Decay:

$$TM \rightarrow TM + TN$$

(c) Neutral Particle Decay:

$$TN \rightarrow TM + TP$$

(d) Positive Particle Decay:

$$TP \rightarrow TP + TN$$

TP and TN represent positively charged and neutral particles respectively. The notation used above is similar to the conventional physics notation. The products of the reactions can themselves undergo the same series of reactions; this can occur an indefinite number of times. The chamber has four fiducials ("X"s) and an identification box.

The descriptions $(\mathfrak{L}(\mathcal{S}))$ are ordered for left-to-right recognition in that the lower left-hand fiducial, FI, appears first and its center is then used as the tail for the descriptions of the rest of the fiducials, FID, the identification box, ID, and the particle tracks PT . The sketches of the primitives are only representative. For example, cm and cp are the names of curves with negative and positive curvature respectively; dp is a short line segment of approximately unit slope. The blank and "don't care" primitives describe known and unknown distances between visible parts of the picture. The primitives eh and ev would be precisely defined a priori since the fiducials are in fixed positions relative to each other. On the other hand, es, the starting primitive would be defined as a search strategy to find the lower arm of the left-corner fiducial.

The use of λ for the vertices of interaction, P and N, illustrate the ability of PDL to deal meaningfully with points as well as edges. Physics pictures of this type are natural candidates for description by recursive syntaxes; the recursive definitions of TM, TP, and TN are based on charge conservation and allow for an indefinite number of well-formed reactions.

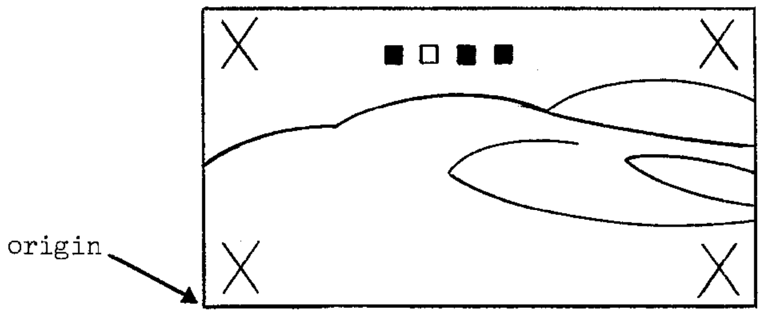


Figure 4.2(a) Sample Picture

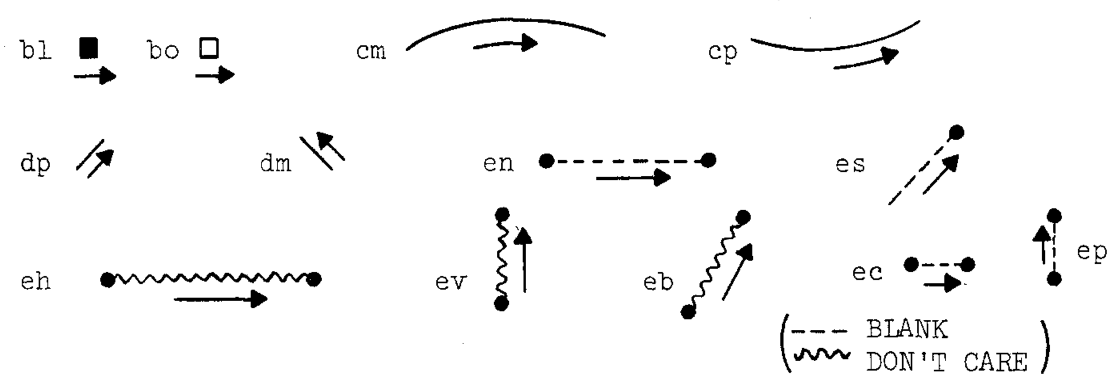


Figure 4.2(b) Primitives

PICTURE	$\rightarrow$	(es + (FI + (FID × (ID × PT))))
FI	$\rightarrow$	(dp + (dm × (dp × (λ - dm))))
FID	$\rightarrow$	((eh + X) + ((ev + X)) - (X + eh)))
ID	$\rightarrow$	((eb + B) + ((ec + B) + ((ec + B) + (ec + B))))
PT	$\rightarrow$	(ep + TM)
X	$\rightarrow$	((dp × dm) × (($\sim$ dp) × (λ - dm)))
B	$\rightarrow$	b0|bl
TM	$\rightarrow$	(cm + MD)|(cm + MP)|cm
MP	$\rightarrow$	(P + ((TM × TP) × TN))|(P + (TM × TP))|(P + TN)
MD	$\rightarrow$	(TM × TN)|TM
TP	$\rightarrow$	(cp + PD)|cp
TN	$\rightarrow$	(en + (N + (TM × TP)))
PD	$\rightarrow$	(TP × TN)|TP
P	$\rightarrow$	λ
N	$\rightarrow$	λ

Figure 4.2(c) Syntax &

Figure 4.2 Particle Physics Example

4.2 KIRSCH'S 45° RIGHT TRIANGLES

Figure 4.3 contains a syntax and examples of two-dimensional 45° right triangles with the same point identifications or labels as that given by Kirsch [1964] (see discussion in section 2.3). The primitives are defined as all translations over a two-dimensional grid of the samples shown. Each point in a triangle is assumed to appear as an "X" on one <u>raster</u> <u>unit</u> (square, grid point).

When a picture is represented as finite grid of points, the possible coordinates of the tail and head of any picture (including λ) are restricted to the grid point coordinates. The definitions of the binary operators as concatenations <u>onto</u> means that the expression $(h + \lambda)$ describes pictures where the coordinates of λ are identical to those of $\text{head}(\alpha)$, $\alpha \in \mathcal{P}(h)$ (the rightmost "X" in α); also, if $\alpha \in \mathcal{P}((h+v))$, α is of the form $\begin{matrix} & X \\ X & X\end{matrix}$. These interpretations are used for digitized pictures.

The identification of the triangle points as interior (I), base (B), hypoteneuse (H), right vertical leg (L), right angle (R), and the vertices bounded by the hypoteneuse (V and W) is accomplished by the rules:

$$\left\{\begin{matrix} I \\ B \\ H \\ L \\ R \\ V \\ W \end{matrix}\right\} \rightarrow \lambda \; .$$

In the examples, the subscript on each "X" indicates its label.

The right-triangle syntax will also generate expressions which do not describe any pictures; this is an example of the problem discussed in 3.3. DH and DI might not be the correct "length" for the * concatenation; if this is the case, as in 3.3, the class of pictures described by the particular T_s is empty.

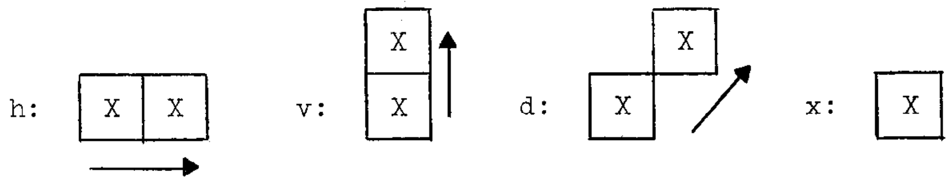


Figure 4.3(a) Primitives

$$
\begin{array}{ll}
\text{RAT} & \rightarrow (((V + h) + (\text{IRAT} + (v + W))) * \text{DH}) \\
\text{IRAT} & \rightarrow (((B + h) + (\text{IRAT} + (v + L))) * \text{DI}) \mid (x + R) \\
\text{DH} & \rightarrow ((d + H) + \text{DH}) \mid d \\
\text{DI} & \rightarrow ((d + I) + \text{DI}) \mid d \\
V & \rightarrow \lambda \\
W & \rightarrow \lambda \\
R & \rightarrow \lambda \\
H & \rightarrow \lambda \\
B & \rightarrow \lambda \\
L & \rightarrow \lambda \\
I & \rightarrow \lambda
\end{array}
$$

Figure 4.3(b) $\mathcal{S}$: $\underline{\text{R}}$ight-$\underline{\text{A}}$ngled $\underline{\text{T}}$riangle

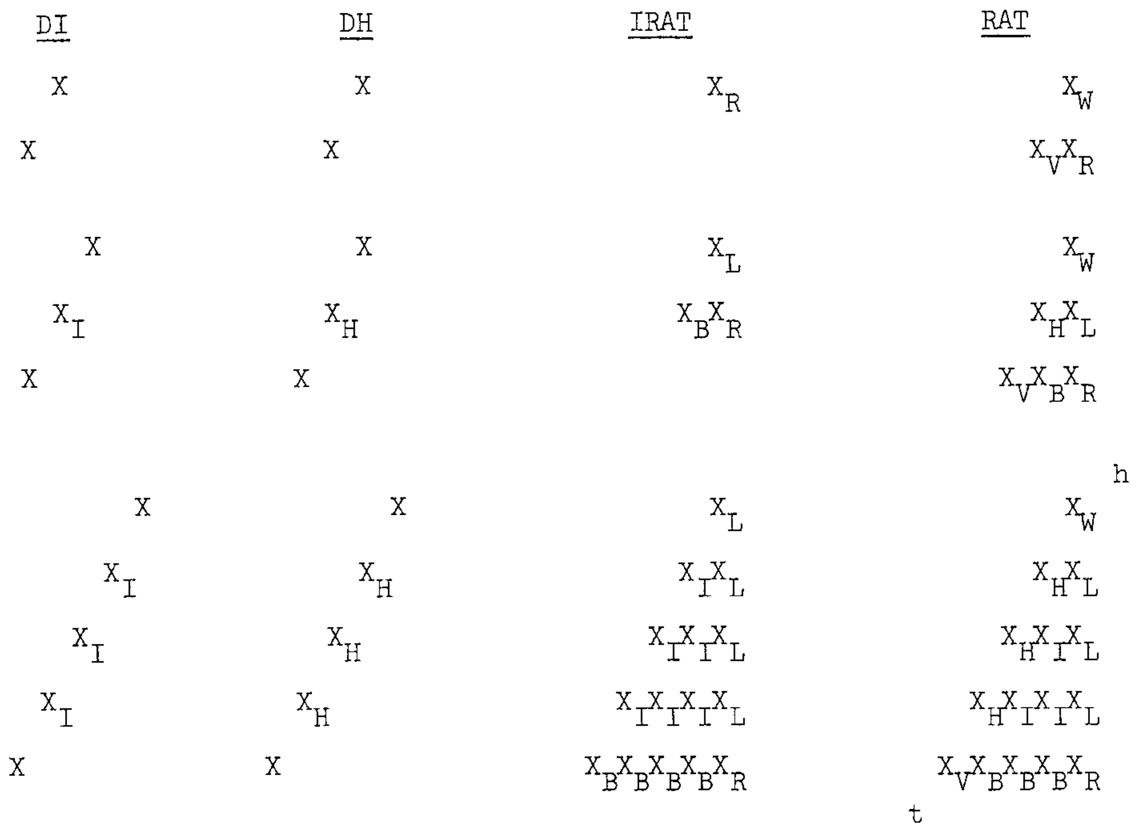


Figure 4.3(c) Examples

Figure 4.3 Right-Angled 45° Triangle of Kirsch

4.3 SIMPLE BLOCK LETTERS AND A PAGE OF ENGLISH TEXT

A block version of the upper case letters of the English alphabet is described in Figure 4.4. Parentheses, which are redundant because of the associativity of the operators, are omitted. The PDL expressions for each letter were formed so that the tail and head is located uniformly throughout the alphabet on the "typographic" line; pictures containing groups of letters and other symbols can then be characterized by PDL

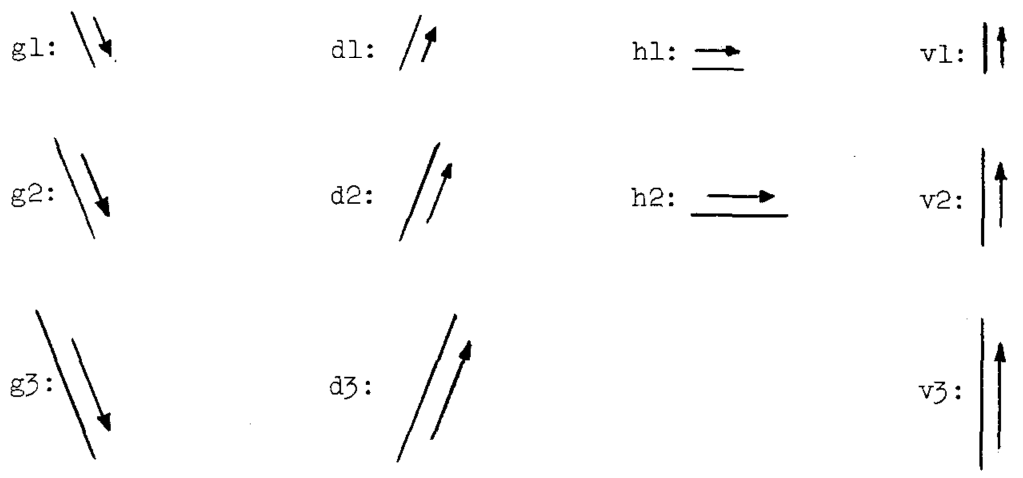


Figure 4.4(a) Primitives

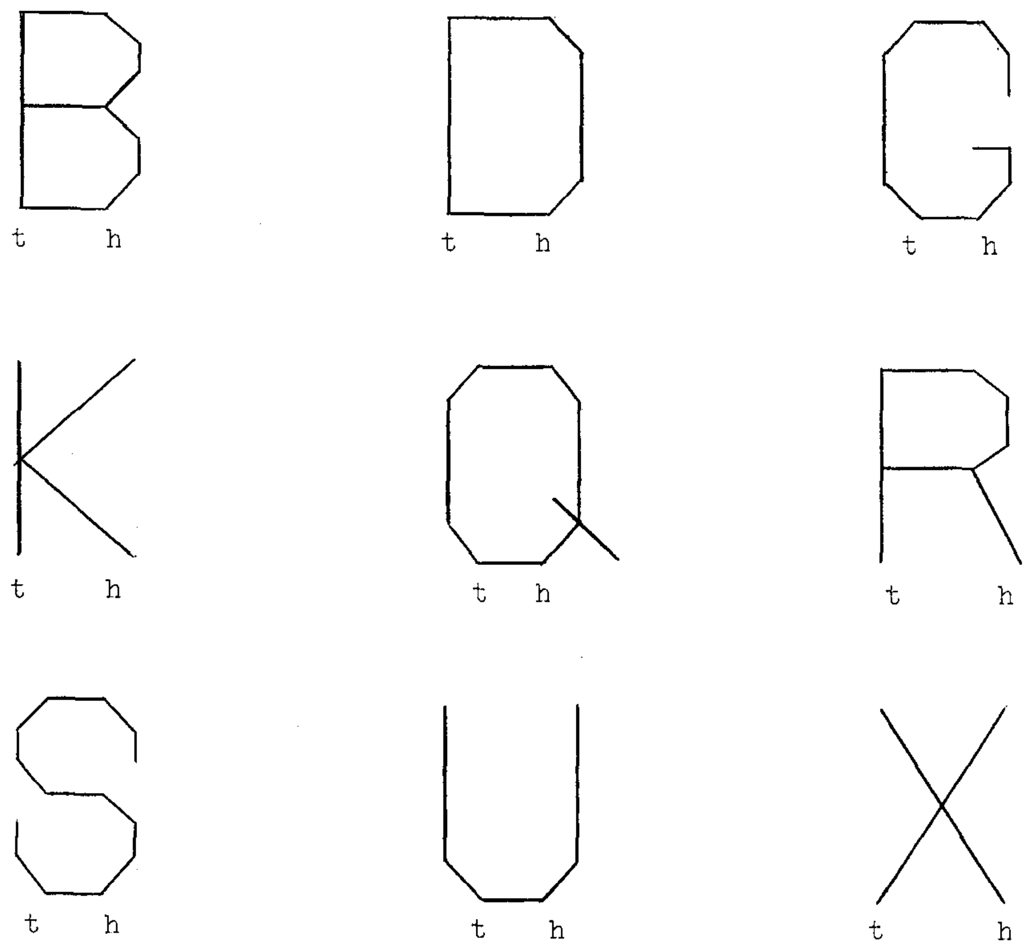


Figure 4.4(b) Examples

$A \rightarrow (d2 + ((d2 + g2) * h2) + g2)$

$B \rightarrow ((v2 + ((v2 + h2 + g1 + (\sim(d1 + v1))) * h2) + g1 + (\sim(d1 + v1))) * h2)$

$C \rightarrow (((\sim g1) + v2 + d1 + h1 + g1 + (\sim v1)) \times (h1 + ((d1 + v1) \times \lambda)))$

$D \rightarrow (h2 * (v3 + h2 + g1 + (\sim(d1 + v2))))$

$E \rightarrow ((v2 + ((v2 + h2) \times h1)) \times h2)$

$F \rightarrow ((v2 + ((v2 + h2) \times h1)) \times \lambda)$

$G \rightarrow (((\sim g1) + v2 + d1 + h1 + g1 + (\sim v1)) \times (h1 + ((d1 + v1 - h1) \times \lambda)))$

$H \rightarrow (v2 + (v2 \times (h2 + (v2 \times (\sim v2)))))$

$I \rightarrow (v3 \times \lambda)$

$J \rightarrow ((((\sim g1) + v1) \times h1) + ((d1 + v3) \times \lambda))$

$K \rightarrow (v2 + (v2 \times d2 \times g2))$

$L \rightarrow (v3 \times h2)$

$M \rightarrow (v3 + g3 + d3 + (\sim v3))$

$N \rightarrow (v3 + g3 + (v3 \times \lambda))$

$O \rightarrow (h1 * ((\sim g1) + v2 + d1 + h1 + g1 + (\sim(d1 + v2))))$

$P \rightarrow ((v2 + ((v2 + h2 + g1 + (\sim(d1 + v1))) * h2)) \times \lambda)$

$Q \rightarrow (h1 * ((\sim g1) + v2 + d1 + h1 + g1 + (\sim(d1 + ((\sim g1) \times g1) + v2))))$

$R \rightarrow (v2 + (h2 * (v2 + h2 + g1 + (\sim(d1 + v1))) + g2)$

$S \rightarrow ((((\sim g1) + v1) \times h1) + ((d1 + v1 + (\sim(g1 + h1 + g1))$
$\quad + v1 + d1 + h1 + g1 + (\sim v1)) \times \lambda))$

$T \rightarrow ((v3 + (h1 \times (\sim h1))) \times \lambda)$

$U \rightarrow ((((\sim g1) + v3) \times h1) + ((d1 + v3) \times \lambda))$

$V \rightarrow ((\sim g3) \times d3 \times \lambda)$

$W \rightarrow (((\sim g3) + d3 + g3) + (d3 \times \lambda))$

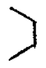
$X \rightarrow (d2 + ((\sim g2) \times d2 \times g2))$

$Y \rightarrow ((v2 + ((\sim g2) \times d2)) \times \lambda)$

$Z \rightarrow ((d3 - h2) \times h2)$

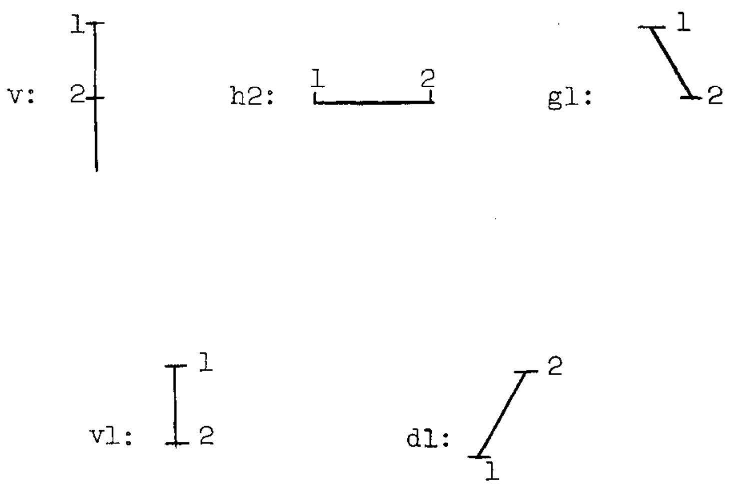
Figure 4.4(c) Primitive Structural Descriptions

Figure 4.4 Simple English Block Characters

easily. The description could be rewritten as a grammar taking advantage of some of the common structures in the letters; for example,  appears in P, R, S, and B . The expressions in Figure 4.4 can be compared to the descriptions used by Narasimhan [1966] (see section 2.3) for generating the upper case English alphabet; because only two nodes of possible concatenation are defined by a PDL description for each picture, it is not necessary to explicitly number nodes and maintain node lists. For a block letter P, Narasimhan's method would lead to the description:

$$\begin{aligned} P \rightarrow (v \cdot (((h2 \cdot g1)[21;\ 1;\ 2] \\ \cdot (v1 \cdot d1)[22;\ 1;\ 1])[21;\ 1;\ 2] \\ \cdot h2)[22;\ 1;\ 1])[11,\ 22] \end{aligned}$$

where the primitives and node labels are:



A page of text is broken into sentences, lines, words, and characters by the syntax of Figure 4.5. Blank primitives establish the connectivity of words on a line (iws), characters within a word (ics), and lines (ils) . left, right and bottom are left, right, and bottom of the page indicators. The PDL expressions of the last figure could be used for the letters generated by CHAR. This type of syntax could conceivably be the basis for analyzing and generating textual information.

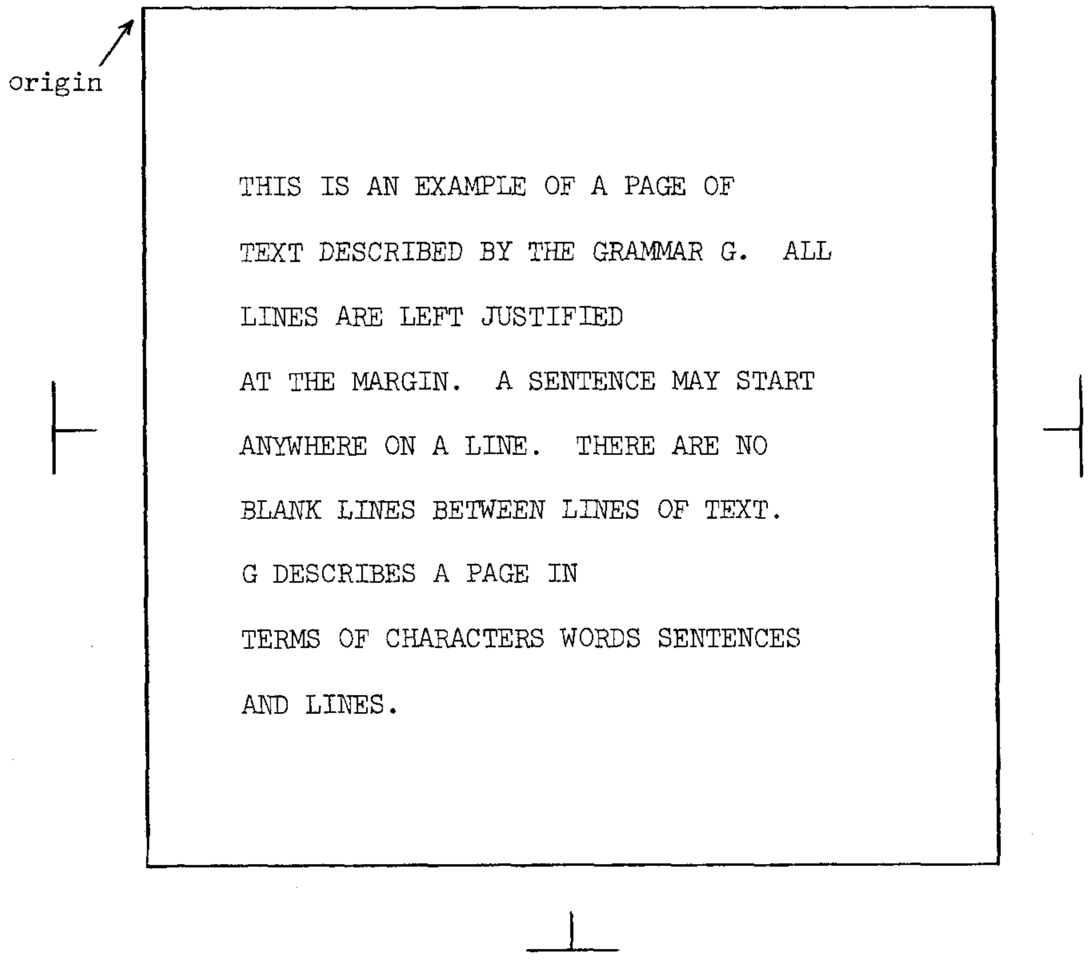

origin
THIS IS AN EXAMPLE OF A PAGE OF
TEXT DESCRIBED BY THE GRAMMAR G. ALL
LINES ARE LEFT JUSTIFIED
AT THE MARGIN. A SENTENCE MAY START
ANYWHERE ON A LINE. THERE ARE NO
BLANK LINES BETWEEN LINES OF TEXT.
G DESCRIBES A PAGE IN
TERMS OF CHARACTERS WORDS SENTENCES
AND LINES.

Figure 4.5(a) Sample Page of Text

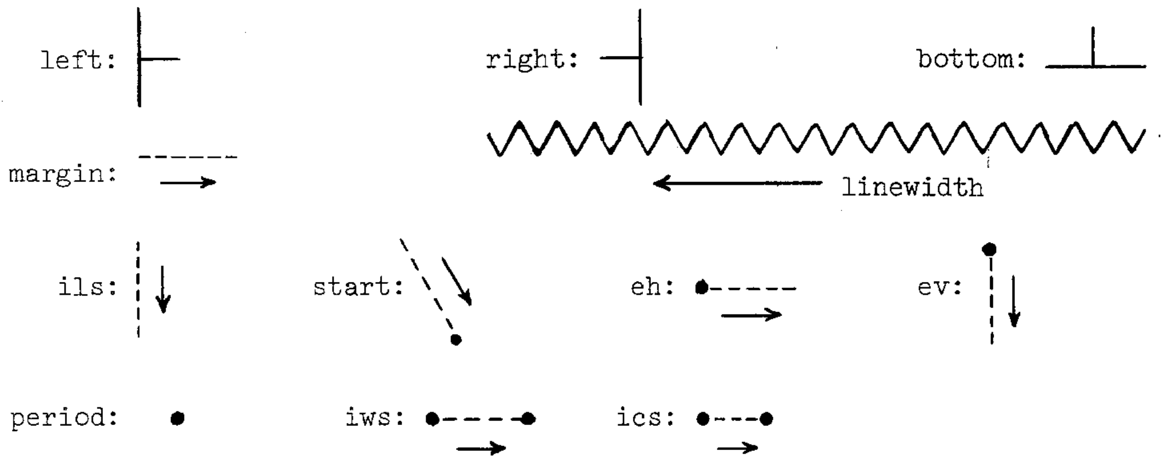


Figure 4.5(b) Primitives

PAGE	→ (start + (S + EOP))
S	→ SENT | (S + SENT)
SENT	→ (BEGINSENT + (L + (ics + period)))
BEGINSENT	→ iws | (EOL + ((linewidth + left) + (ils + margin))) | λ
L	→ LINE | (L × ((ils + margin) + LINE))
LINE	→ WORD | (LINE + (iws + WORD))
WORD	→ CHAR | (WORD + (ics + CHAR))
CHAR	→ A | B | C .... | Y | Z
EOP	→ (ev + bottom)
EOL	→ (eh + right)

Figure 4.5(c) $\mathcal{G}$: Syntax For a Page of Text

Figure 4.5 A Page of English Text

4.4 CLOSED BOUNDARIES OF FIGURES

A description of the edge sequences comprising the boundary of a figure can be easily expressed by a PDL grammar.

Example:

$$\text{BOUNDARY} \rightarrow (\text{CURVELIST} * \lambda)$$
$$\text{CURVELIST} \rightarrow \text{CURVE} \mid (\text{CURVELIST} + \text{CURVE})$$
$$\text{CURVE} \rightarrow c1 \mid c2 \mid \;\ldots\ldots\; \mid cn\ ,$$

where $\{ci,\ i = 1,\ n\}$ is the set of edge or curve types that may appear in the figure.

The chromosome syntax given by Ledley et. al [1965] (see section 2.3) could be directly rewritten as a PDL syntax by inserting + operators, appropriate parentheses, and the final $* \lambda$; the samples of Figure 2.3 would then be:

$$\langle \text{arm} \rangle \rightarrow (B + \langle \text{arm} \rangle) \mid (\langle \text{arm} \rangle + B \mid A$$
$$\langle \text{side} \rangle \rightarrow (B + \langle \text{side} \rangle) \mid (\langle \text{side} \rangle + B) \mid B \mid D$$
$$\langle \text{submedian chromosome} \rangle \rightarrow ((\langle \text{arm pair} \rangle + \langle \text{armpair} \rangle) * \lambda)$$

4.5 FLOW CHARTS

None of the previous examples require the use of label designators and the / operator. Flow charts provide a good illustration of a practical picture class for which the set $\{+, -, \times, *, \sim\}$ is not adequate. A notational convenience is introduced for the examples of this section:

Consider a PDL expression with standard form,

$$S = S(p_1^{\ell_1}, p_2^{\ell_2}, \ldots, p_n^{\ell_n}, p_{n+1}, p_{n+2}, \ldots, p_m),$$

where $p_i^{\ell_i}$, $1 \leq i \leq n$ are the labeled primitives of the form and p_i, $n < i \leq m$ are the primitives without labels. Then $S^{\underline{\ell}} \equiv S(p_1^{\ell_1 \ell}, p_2^{\ell_2 \ell}, \ldots, p_n^{\ell_n \ell}, p_{n+1}, p_{n+2}, \ldots, p_m)$; that is, the underbar on a label means that all primitives already labeled in the standard form of the expression, and only those, are given the additional label, e.g., $(a^i + b)^{\underline{j}} \equiv (a^{ij} + b)$. This eliminates many redundant labels that would otherwise appear in the standard form of the PDL descriptions generated by the flow chart syntaxes.

In a recent paper (Böhm and Jacopini [1966]), Jacopini presents a number of "base" diagrams into which a large class of flow charts may be decomposed. The purpose here is not to pursue Jacopini's theoretical results, but to provide a structural description of this class of flow charts in terms of the base diagrams and their components. Figure 4.6(a) contains samples of the primitives. The line segments with arrow heads leading from enter, fn, and cond may be any sequence of concatenated segments thus allowing the head of these primitives to be placed anywhere in a picture relative to the tail. The box in fn is a functional box and pred represents a predicate or test. cond may be either the true or false branch of the predicate; the initial blank part at its tail carries it to one of the vertices of the diamond of pred . The primitives can be further described in PDL as concatenations of line segments and circles, cond could be given a true or a false label, and character strings could be defined within the boxes.

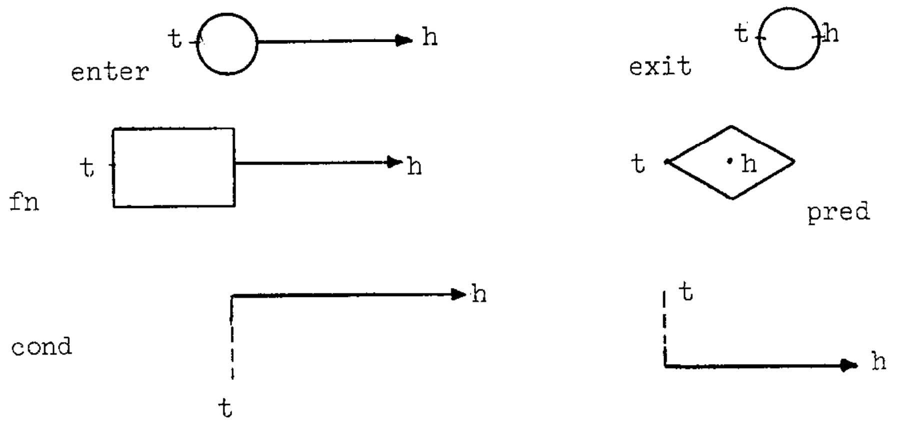


Figure 4.6(a) Primitives

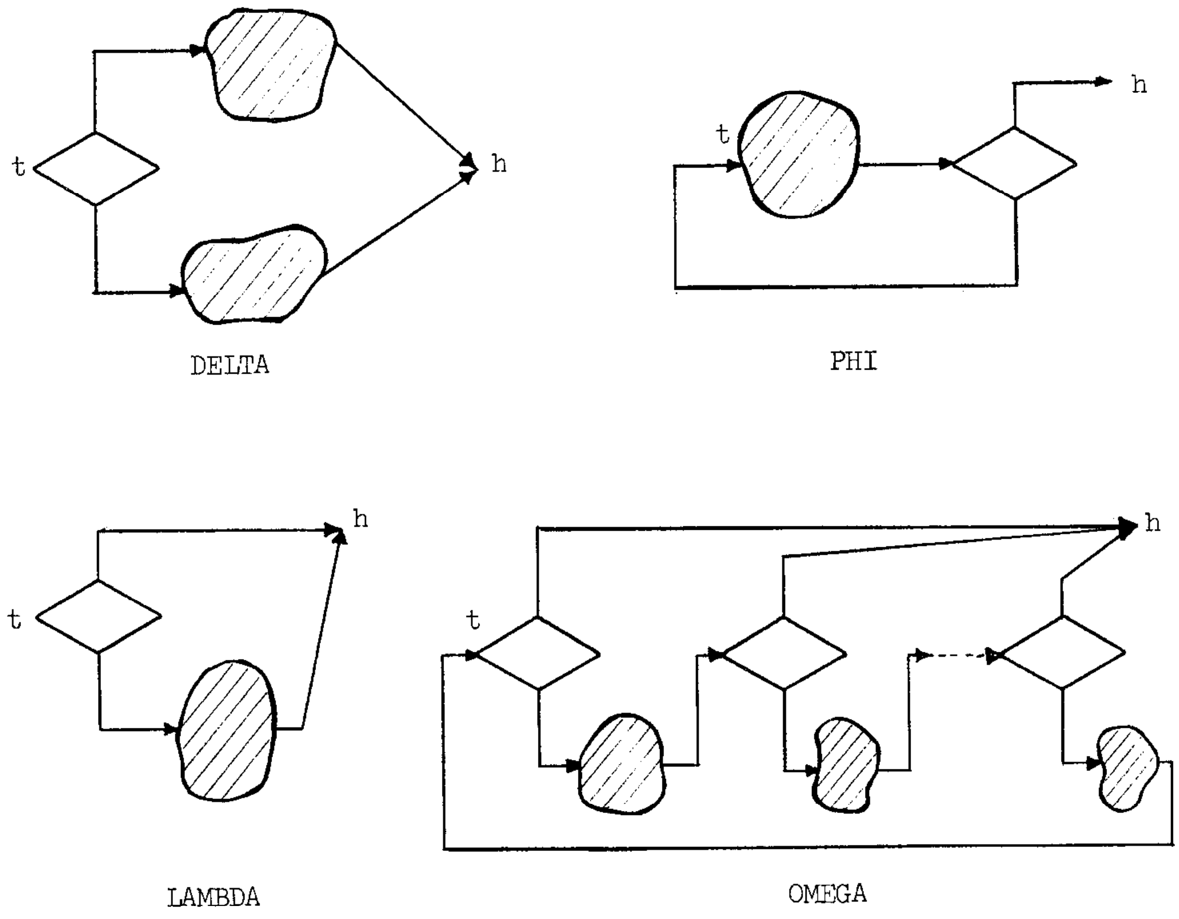


Figure 4.6(b) Diagrams

FLOWCHART	$\rightarrow$	$(\mathrm{enter} + (\mathrm{FC} + \mathrm{exit}))$
FC	$\rightarrow$	$\mathrm{JACOPINI} \mid (\mathrm{JACOPINI} + \mathrm{FC}^{\underline{fc}})$
JACOPINI	$\rightarrow$	$\mathrm{DELTA} \mid \mathrm{PHI} \mid \mathrm{LAMBDA} \mid \mathrm{OMEGA} \mid \mathrm{fn} \mid \lambda$
DELTA	$\rightarrow$	$(\mathrm{pred} + ((\mathrm{cond} + \mathrm{FC}^{\underline{d1}}) * (\mathrm{cond} + \mathrm{FC}^{\underline{d2}})))$
PHI	$\rightarrow$	$(((\mathrm{FC}^{\underline{p}} + \mathrm{pred}) * (\sim \mathrm{cond})) + \mathrm{cond})$
LAMBDA	$\rightarrow$	$(\mathrm{pred} + ((\mathrm{cond} + \mathrm{FC}^{\underline{\ell}}) * \mathrm{cond}))$
OMEGA	$\rightarrow$	$((((\mathrm{pred}^{i} + \mathrm{cond}) + (\mathrm{FC}^{\underline{m1}} + \mathrm{OMD})) * \lambda)$
		$\times ((/\mathrm{pred}^{i}) + (\mathrm{cond} + \lambda^{i})))$
OMD	$\rightarrow$	$\lambda \mid (\mathrm{pred} + ((\mathrm{cond} + (/\lambda^{i})) \times (\mathrm{cond} + \mathrm{FC}^{\underline{m2}}))) \mid$
		$(\mathrm{OMD}^{\underline{m3}} + (\mathrm{pred} + ((\mathrm{cond} + (/\lambda^{i})) \times (\mathrm{cond} + \mathrm{FC}^{\underline{m2}}))))$

Figure 4.6(c) Flowchart Syntax

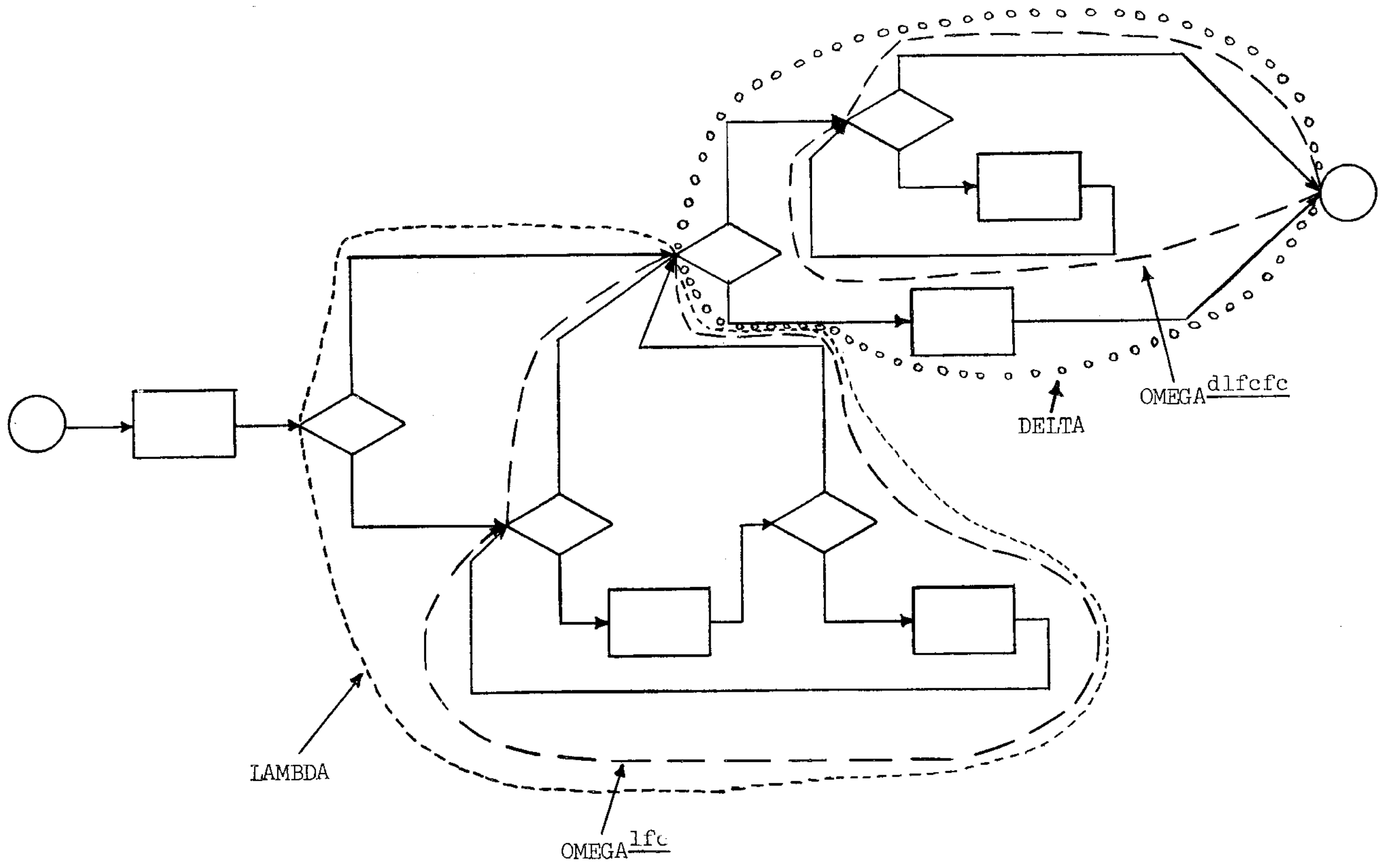


Figure 4.6(d) Sample Flow Chart

The base diagrams of Figure 4.6(b) are slight modifications of those of Jacopini; the primitives fn and λ will also be considered as base diagrams. The hatched areas denote any base diagram or series of them.

The syntax of Figure 4.6(c) generates descriptions of <u>all</u> flow charts formed by the concatenation and composition of elements of {DELTA, PHI, LAMBDA, OMEGA, fn, λ}, and <u>no</u> other; enter and exit are included for completeness. An informal proof of this statement proceeds as follows:

1. The syntaxes for DELTA, PHI, LAMBDA, and OMEGA generate descriptions of each of their respective base diagrams; this may be verified by drawing the graph of each expression by the methods of section 3.4.2.
2. The recursions for each base diagram allow arbitrary composition of diagrams.
3. The expressions generated by FC denote an indefinite series of these diagrams.
4. With the enter and exit, 1, 2, and 3 above are the only expressions generated by FLOWCHART .
5. The use of different label designators for FC wherever it appears in the syntax ensures that the labels on λ and pred in OMEGA and OMD will be unique for each generation of the OMEGA diagram. For example, FC could generate:

   $$(\mathrm{JACOPINI} + (\mathrm{JACOPINI} + (\mathrm{JACOPINI} + \mathrm{JACOPINI}^{\underline{fc}})^{\underline{fc}})^{\underline{fc}})$$

   which is equivalent to:

$$(\mathrm{JACOPINI} + (\mathrm{JACOPINI}^{\underline{fc}} + (\mathrm{JACOPINI}^{\underline{fcfc}} + \mathrm{JACOPINI}^{\underline{fcfcfc}})))$$

This is illustrated in the example of Figure 4.6(d), where the PDL expression for the OMEGA diagram in LAMBDA contains λ^{ilfc} and $pred^{ilfc}$, and that for the OMEGA in DELTA contains $\lambda^{idlfcfc}$ and $pred^{idlfcfc}$.

The second flow chart example shows how an algorithmic programming language can be defined by the syntax of its flow charts in conjunction with the syntax of its strings; the latter describes the allowable strings in the language while the former denotes flow of control. The primitives are identical with those of the last example. Figure 4.7(a) contains a partial syntax for a simple ALGOL-like language. ASSIGNMENT statements, BLOCKHEAD (e.g., begin⟨declaration list⟩), BLOCKTAIL (e.g., end), AE (⟨arithmetic expression⟩), BE (⟨Boolean expression⟩), and VARIABLE are not defined further since this would add nothing essential to the example. The various statement types are similar to a subset of ALGOL 60 (Naur et. al. [1963]). GO TO statements are not included; they cannot be translated, from their syntax alone, into a flow chart.

With the exception of ASSIGN, INIT, INC, and TEST, there is a one-to-one correspondence between the elements of the flow chart syntax of Figure 4.7(b) and the language syntax; each component of the language translates into a flow chart component. Examples of the flow chart elements are given in Figure 4.7(c); unlabeled hatched areas may contain any diagram generated by STMNT . The same technique as the last example guarantees unique labels for ASSIGN and TEST in the FOR diagram.

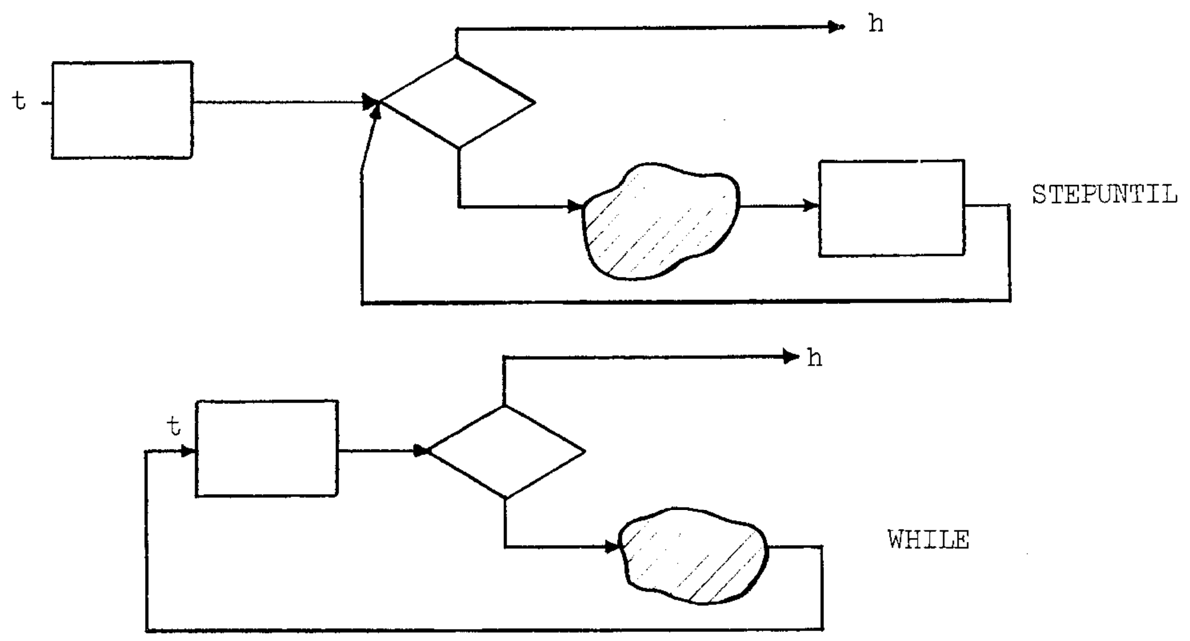
Several automatic flow chart generating programs have been developed. Some of these require the programmer to insert detailed flow charting instructions or comments in his source code, (e.g., Knuth [1963]).

PROGRAM	→	BLOCK
BLOCK	→	BLOCKHEAD;STATEMENTLIST BLOCKTAIL
STATEMENTLIST	→	STATEMENT|STATEMENT; STATEMENTLIST
STATEMENT	→	BASIC|CONDITIONAL
BASIC	→	ASSIGNMENT|FOR|BLOCK
CONDITIONAL	→	IFTHEN|IFTHENELSE
FOR	→	STEPUNTIL|WHILE
STEPUNTIL	→	for VARIABLE := AE step AE until AE do STATEMENT
WHILE	→	for VARIABLE := AE while BE do STATEMENT
IFTHEN	→	if BE then BASIC
IFTHENELSE	→	if BE then BASIC else STATEMENT

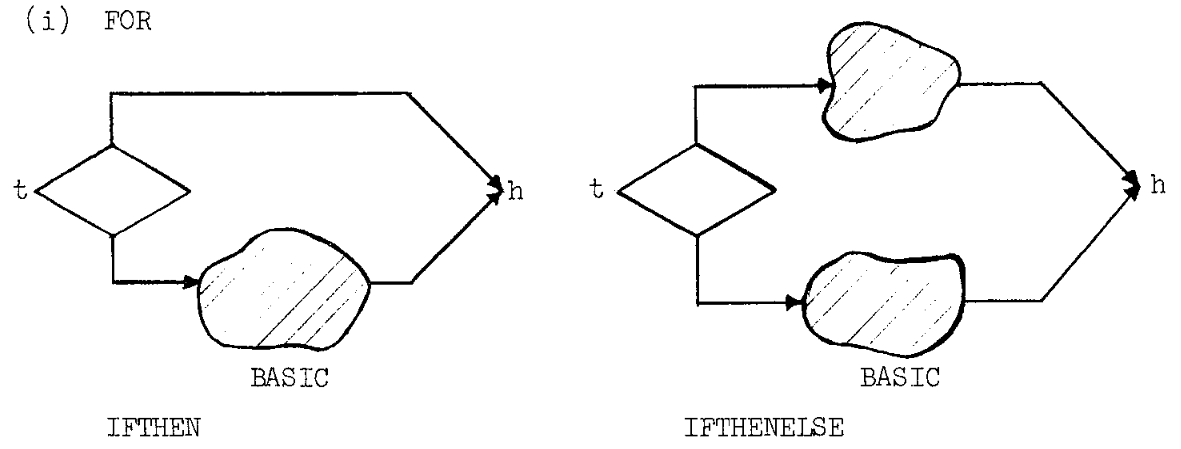
Figure 4.7(a) Small Language Syntax

PROGRAM	→	BLOCK
BLOCK	→	(entry + (S + exit))
S	→	STMNT|(STMNT + $S^{\underline{s}}$)
STMNT	→	BASIC|CNDTNL
BASIC	→	ASSIGN|FOR|$\text{BLOCK}^{\underline{b}}$
CNDTNL	→	IFTHEN|IFTHENELSE
ASSIGN	→	fn
FOR	→	STEPUNTIL|WHILE
STEPUNTIL	→	(INIT + ((((TEST^{su} + cond) + $\text{STMNT}^{\underline{su}}$)
		* (~ INC)) × ((/TEST^{su}) + cond)))
WHILE	→	((((ASSIGN + TEST)^{w} + cond) * (~ $\text{STMNT}^{\underline{w}}$))
		× ((/(ASSIGN + TEST)^{w}) + cond))
IFTHEN	→	(pred + ((cond + $\text{BASIC}^{\underline{i}}$) * cond))
IFTHENELSE	→	(pred + ((cond + $\text{BASIC}^{\underline{it1}}$) * (cond + $\text{STMNT}^{\underline{it2}}$)))
INIT	→	fn
INC	→	fn
TEST	→	pred

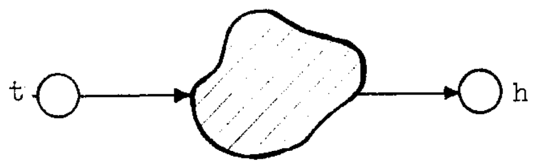
Figure 4.7(b) Small Language Flow Chart Syntax



(i) FOR



(ii) CNDTNL



(iii) PROGRAM or BLOCK

Figure 4.7(c) Examples

Figure 4.7 String and Flow Chart Syntax for a Small Algorithmic Language

Sherman [1966] describes the control syntax of a source language by a series of descriptors, which are then used to produce a general flow charting program for the language; however, Sherman is unable to handle languages with recursively-defined elements. The methods presented in the last example could serve as the basis for a general flow-charting program, which does not have the above restrictions. Sutherland [1966] has designed and implemented a system for graphically specifying programs (on a computer-controlled display) and executing them; he uses an unconventional set of primitive elements for the flow charts and the computations. Flow charts could be drawn, syntactically analyzed, and executed within the PDL system to provide a more conventional and natural system of this type; a suitable set of semantic rules $\mathcal{J}_m$ would have to be designed along with the interactive components of the system. Finally, it might be simpler for a compiler to deal with the derived flow chart rather than the source program for generating efficient code.

It should be noted that the above applications are only educated predictions by this writer, since the details of such PDL flow chart generation and analysis systems have not been worked out.

4.6 SOME DESCRIPTION LIMITATIONS OF THE PDL SYSTEM

PDL is not a description panacea; the previous examples suggest its range of application. Further experimentation is necessary in order to precisely delimit the class of pictures for which useful descriptions may be obtained. At this stage, nevertheless, it is possible to enumerate some of its limitations and some possible extensions.

The class of PDL descriptions, $\mathcal{L}(\mathcal{G})$, that may be generated from a context free grammar $\mathcal{G}$ is theoretically limited (Chomsky [1959], Ginsburg [1966]). Consider the description of an arbitrary "staircase" of "X"'s on a grid (Figure 4.8(a)). If the notation $\sum_{i=1}^{n} a$ represents

$$\underbrace{a + a + \ldots + a}_{n\ a\text{'s}} ,$$

and the primitives h and v are those of Figure 4.2, then the set

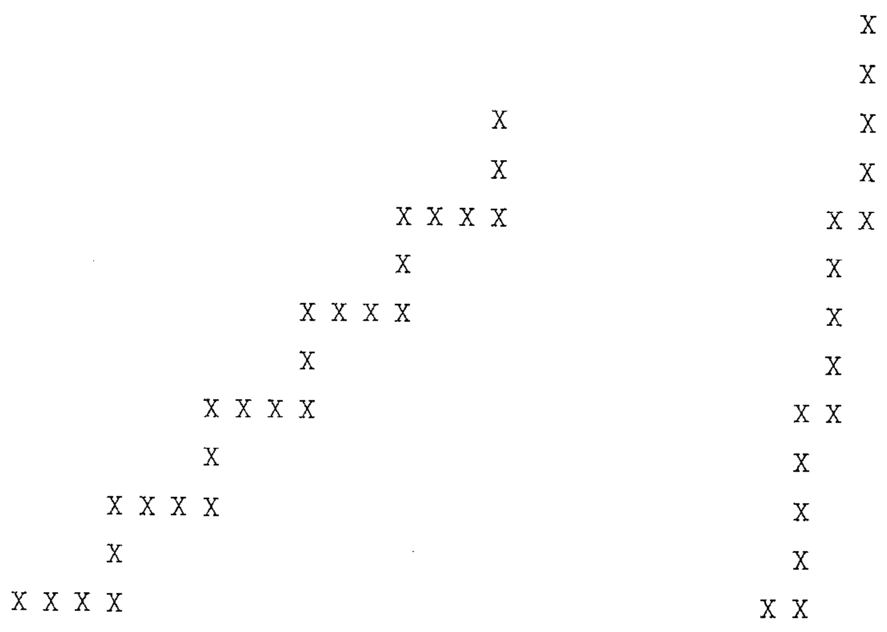
$$\{(\sum_{i=1}^{\ell} [\sum_{j=1}^{m} h + \sum_{k=1}^{n} v]) | \ell,\ m,\ n \geq 1\},$$

where "[" and "]" indicate expression grouping, contains all possible PDL staircase descriptions (without redundant parentheses) with constant horizontal and vertical distances. This set, however, cannot be generated by a context-free grammar since this would imply that

$$\{((ab)^{m}(cd)^{n})^{\ell} | m,\ n,\ \ell \geq 1\}$$

is a context-free language.

Concatenation of picture elements is the only explicit relation in PDL. The use of blank primitives allows many simple geometric relations among disjoint picture elements to be expressed. There are a great many other relations that one might like to see directly expressible in a picture language. For example, $\mathcal{P}((S_1 + (b + S_2))$, where S_1 and S_2 describe picture components and b is a blank primitive, might be the class of pictures such that the elements of some subset of $\mathcal{P}(S_1)$ is contained within those of a subset of $\mathcal{P}(S_2)$; alternatively, one might



$$\left(\sum_{i=1}^{5} \left[\sum_{j=1}^{3} h + \sum_{k=1}^{2} v \right] \right) \qquad \left(\sum_{i=1}^{3} \left[\sum_{j=1}^{1} h + \sum_{k=1}^{4} v \right] \right)$$

Figure 4.8(a) Staircase of "X"'s

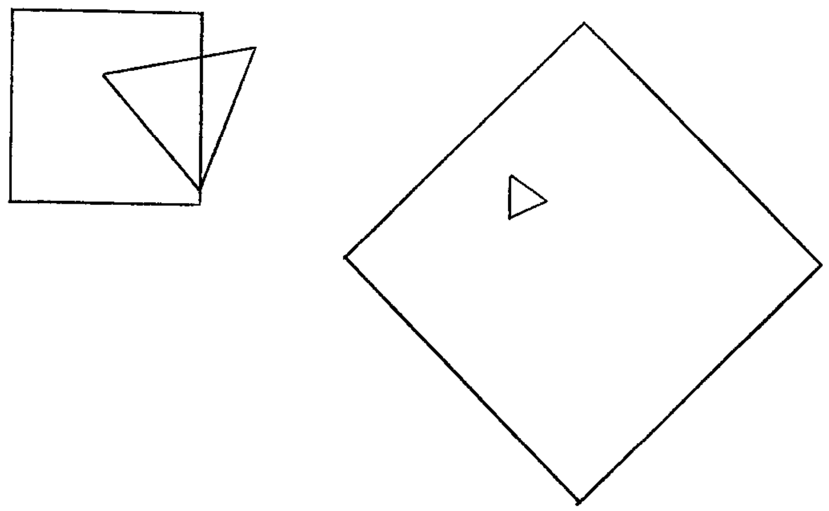


Figure 4.8(b) Complex Relations Among Figures

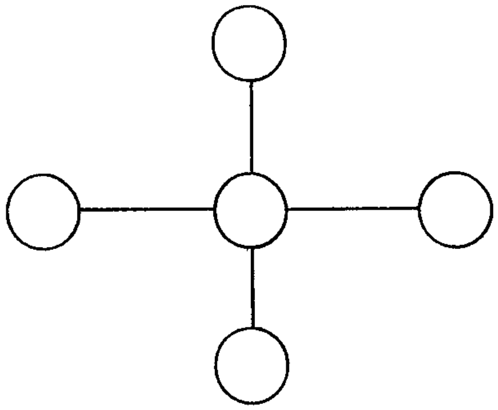


Figure 4.8(c) More Than Two Concatenation Points on a Primitive

Figure 4.8 Some Description Limitations of the PDL System

want to say that some elements of $\mathcal{P}(S_1)$ overlap those of $\mathcal{P}(S_2)$, (Figure 4.8(b)). In either case, the intended relation is not expressible generally; if $\mathcal{P}(S_1)$ and $\mathcal{P}(S_2)$ are severely restricted and b defined appropriately, then $(S_1 + (b + S_2))$ might be satisfactory, but the more complex relation is not obtious. A related difficulty is that of relations depending on magnifications, rotations, and other transformations of pictures. In Figure 4.8(b), it might be desired to group the small triangle with the small square; if a wide range of sizes and rotations of these elements are possible, then a PDL description reflecting this size grouping cannot be found.

Each primitive is restricted to only two points of possible concatenation. There are many cases where more than two concatenation points appear to be necessary. Some of these can be treated in a natural manner by a judicious choice of the tail and head. In Figure 4.8(c), the circles c and line segments ℓ are primitives; c has both its tail and head at the center of the circle. The multiple concatenation of the lines onto the central circle can be expressed by adjoining blank primitives b to each end of a line segment; then a description is:

$$P \to (c + ((L + c) \times (L + c) \times (L + c) \times (L + c)))$$
$$L \to (b + \ell + b)$$

One possible generalization of PDL to handle the above problems can be suggested. A suitable extension of Anderson's predicates [1967] (see section 2.3) would enable complex relations among picture components to be described; these could be in the form of additions to each syntax rule. Use could be made of the preservation of the topological relations among picture components under a large class of transformations. For example, if the primitive structural description of a picture α is:

$$T_s(\alpha) = S(p_1, p_2, \ldots, p_n)$$

$$T_v(\alpha) = (D(\beta_1), D(\beta_2), \ldots, D(\beta_n),$$

where

$$\beta_i \in \mathcal{P}(p_i), \qquad i = 1, \ldots, n$$

and A represents a magnification or rotation transformation, then

$$T_s(A\alpha) = S(q_1, q_2, \ldots, q_n)$$

$$T_v(A\alpha) = (D(A\beta_1), D(A\beta_2), \ldots, D(A\beta_n)),$$

where

$$A\beta_i \in \mathcal{P}(q_i) \ .$$

These suggestions are left for future work. The important points are:

1. a large, interesting, and useful class of pictures can be described in a simple and natural manner within the PDL system, and
2. the system is capable of extension without destroying its basic simplicity.

CHAPTER 5

PICTURE PARSING

5.1 THE ANALYSIS PROBLEM

The basic information required for the analysis of a picture class is:

(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 $\alpha_i \in \mathcal{P}_{\mathcal{G}}$, $i = 1, \ldots, n$; 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 subset 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 $D(\beta_i)$). The entire analysis process is directed by the PDL description of the picture class 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, possibly based on the receptor/categorizer model.

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 PDL 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 [1964] -- 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 $\mathcal{L}(\mathcal{S})$ 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 the output of a generation procedure. Since, in general, many PDL descriptions are possible for the same picture, a string analysis based on $\mathcal{S}$ would often fail, even if the picture were in $P_{\mathcal{S}}$. However, a PDL expression can be transformed into a graph or a list of primitives and the recognition treated as in number 2 or 3 above. (Carlbom [1967] has written a program that transforms PDL expressions into a primitive connection matrix.)

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 will be the main emphasis. One of the major advantages of this approach to analysis is that the primitives in a digitized picture can be recognized more easily than in ad hoc methods.

5.2 GOAL-ORIENTED PICTURE PARSING

String language analyzers that employ the syntax explicitly are usually called <u>syntax-directed</u> (Floyd [1964], Shaw [1966b], and Feldman and Gries [1967] contain surveys of typical syntax-directed compiling methods). Many of the same techniques are used for picture parsing. There are several important differences between the recognition of one-dimensional strings and two/three-dimensional pictures. These differences lead to the selection of a goal-criented analysis scheme.

5.2.1 BOTTOM-UP AND TOP-DOWN ANALYZERS

5.2.1.1 SYNTAX ANALYSIS OF STRING LANGUAGES

Assume that P is the distinguished symbol of a grammar $\mathcal{G}$. A <u>bottom-up</u> 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 <u>top-down</u> 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 $s \in \mathcal{L}(\mathcal{G})$ and $\mathcal{G}$ is unambiguous; Figure 5.1 illustrates the tree formation for both types. Most syntax-directed compilers use a combination of these.

G: P → (start + TRACK)
TRACK → beam | (neg + PRS)
PRS → (PAIR + PRS) | PAIR
PAIR → (pos X neg)

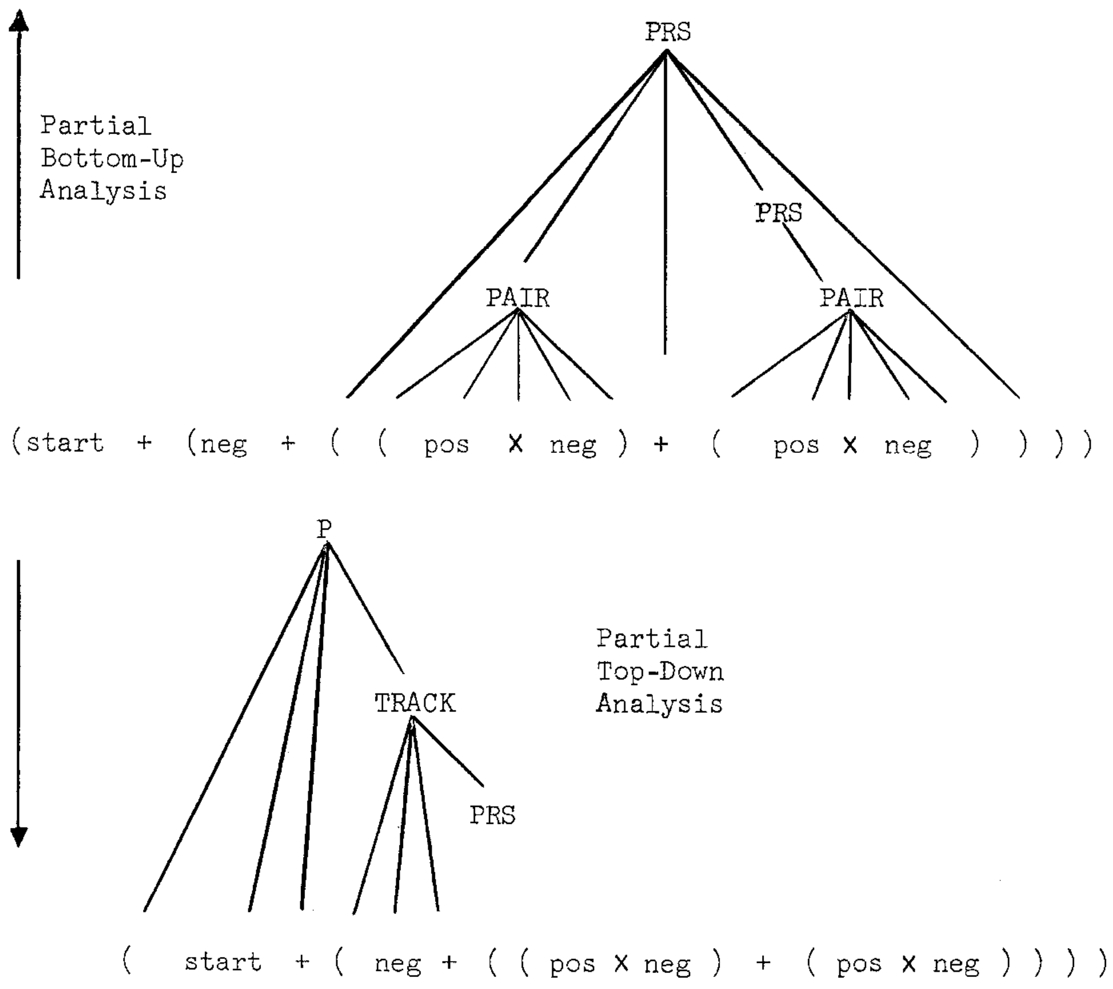

Partial
Bottom-Up
Analysis
PRS
PRS
PAIR
PAIR
(start + (neg + ((pos X neg) + (pos X neg))))
P
TRACK
PRS
Partial
Top-Down
Analysis
(start + (neg + ((pos X neg) + (pos X neg))))

Figure 5.1 Top-Down and Bottom-Up String Analysis

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 by an input routine, say GETNEXTSYMBOL(loc), where loc is an index that points 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 5.1 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.

5.2.1.2 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 analogous to the recognition of terminal symbols by the input devices of computers - and group them into the structures described in $\mathcal{G}$.

Given a grammar $\mathcal{G}$ 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 $\mathcal{G}$. (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 almost impossible. 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 as in possibilities 2, 3, or 4 of section 5.1; however, this would be pushing the most difficult problem outside of the parsing system.)

A pure top-down or goal-oriented analyzer starts with P and attempts to generate from left to right, a sentence $S \in \mathfrak{L}(\mathcal{G})$ such that $S = T_s(\alpha)$. 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 $\wp$(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 5.1 illustrate the approach; the method for string analysis is essentially that of Leavenworth [1964]. The propositional connectives $\wedge$ ("and") and $\vee$ ("or") are to be interpreted from left-to-right in the McCarthy sense (McCarthy [1963]); i.e. $A \wedge B$ means _if_ A _then_ B _else_ _false_ and $A \vee B$ means _if_ A _then_ _true_ _else_ B .

1. _Top-Down String Parser for $\mathcal{S}$ of Figure 5.1_

```
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('X') ∧ Lfs('neg') ∧ Lfs(')');
```

Lfs(name) returns true if the next symbol in the output string is name, and false otherwise. The string pointer and its administation is omitted.

2. Top-Down Picture Parser for $\mathcal{G}$ of Figure 5.1

```
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,hn) ∧ PRS(hn,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,hl) ∧ 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. The parser is executed by the call: P(org,Ω), where org is the picture origin and Ω denotes undefined. This example omits several essential features that must be incorporated when the full PDL operator set is used in the syntax; these are included in the general parser.

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

1. The language portion of the analysis (stepping through the grammar $\mathcal{G}$) is conceptually very simple.
2. The syntax directly expresses the algorithm for analysis.
3. It was conjectured (and verified later) 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 primitive may be used later in the analysis without re-recognition.
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.

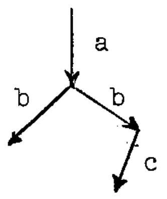
As well as the use of two or three-dimensional tail and head pointers, there are a number of other interesting differences between string and picture parsing that must be taken into account in a general parser:

1. Multiple Recognition

Consider a picture search for primitives that satisfy the expression: $(a + (b \times b))$. The first element of $\wp(b)$ may be recognized 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:

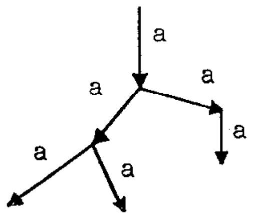


If the first b found is the one on the left branch of the tree, then the parse would try to find a c adjoined to it and fail. One solution is to change the strictly left-to-right recognition if a failure occurs in a commutative expression; in the above example, the search could back-up after the failure 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:

$$A \rightarrow (B * C) | (C * B)$$
$$D \rightarrow (E \times F) | ((F \times E)^{\ell} + (/((\sim E) + F)^{\ell})))$$

3. Detection

A similar difficulty can occur in a detection problem. If the parse is looking for (a + a + (a x a)) in a picture whose primitives have the tree:



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 number 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.

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 $\wp(a)$ regardless of whether more a's were concatenated onto it. This problem disappears for simple recursions if they appear as the first alternative of a production.

5. The effect of - and ~

Since the expression $(A - B)$ can be rewritten as $(A + ((\sim B) \times \lambda))$, 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 $(a + (\sim (b + c))) \equiv (a + ((\sim c) + (\sim b)))$ and use the latter. This could be done more easily by transforming the grammar before starting the parse so that $\sim$ only applies 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 (Irons [1961], Warshall [1961]). 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 the implemented system is based.

5.3 A GENERAL PICTURE PARSING ALGORITHM

Each production of the grammar is assumed to have one of the forms:

$$A \rightarrow (B \phi_b C)$$

$$A \rightarrow (B \phi_b C)^{\ell}$$

$$A \rightarrow D$$

$$A \rightarrow (/p)$$

$$A \rightarrow (\sim p)$$

$$A \rightarrow (\sim (/p))$$

where $\phi_b \in \{+, \times, *\}$, 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), p is a primitive class name with or without a label, and ℓ is a label. This will be called the PDL standard form of the grammar.

A grammar can be put into standard form by employing the algebraic properties of the operators and adding productions.

Examples:

1. $\mathcal{G}$: $A \rightarrow (\sim B)$

   $B \rightarrow (c + d)$

   $\mathcal{G}_{sf}$: $A \rightarrow BM$

   $BM \rightarrow (D + C)$

   $D \rightarrow (\sim d)$

   $C \rightarrow (\sim c)$

where $\mathcal{G}_{sf}$ is the standard form of $\mathcal{G}$.

2. $\mathcal{G}$: $A \rightarrow ((b + c) - d)$ $\mathcal{G}_{sf}$: $A \rightarrow (B + C)$

$B \rightarrow (B1 + C1)$

$C \rightarrow (D \times L)$

$D \rightarrow (\sim d)$

$B1 \rightarrow b$

$C1 \rightarrow c$

$L \rightarrow \lambda$

3. $\mathcal{G}$: $A \rightarrow (b + (\sim A))|b$ $\mathcal{G}_{sf}$: $A \rightarrow (B + C)|b$

$C \rightarrow (A + D)|(\sim b)$

$D \rightarrow (\sim b)$

$B \rightarrow b$

The implemented system mechanically performs these transformations for a subset of PDL; by maintaining appropriate pointers, a parse can be exhibited in terms of the original grammar. It is believed, but not formally verified, that a general grammar incorporating all features of PDL can be transformed mechanically into standard form and that the transformation is reversible; in this case, no generality is lost by solving the parsing problem for grammars in PDL standard form.

The difficulties mentioned in the last section are handled in the obvious 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 $\sim$ operators can no longer occur when the grammar is in standard form.

Two additional assumptions are made for convenience:

1. Only vPDL's are generated by $\mathcal{G}$.
2. Left-recursive productions are not allowed. This is to be interpreted as left recursion in the picture sense, not the language 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 (Cheatham and Sattley [1964]); generality is maintained since these productions may be replaced by right-recursive ones.

The following algorithm will perform a goal-oriented parse of a picture for any grammar in PDL standard form that has the above properties:

```
Boolean Procedure  Parse(L, rpn, t, h);
if rpn > Numberofalternatives(L) then Parse := false
else
begin
    R := Rightpart(L, rpn);
    if Prim(R)  then  bool := Lookforprimitive(R, t, h)
    else
    if  Nonterminal(R)  then  bool := Parse(R, 1, t, h)
    else
    begin
        bool := false;
        hl := hdl := if  Catop(R) = '*' ∧ h ≠ Ω  then  h  else  Ω;
        leftbool := Parse(Leftop(R), 1, t, hdl);
        while ¬ bool ∧ leftbool do
        begin
            hdr := h;
            if  Catop(R) = '+'  then
                bool := Parse(Rightop(R), 1, hdl, hdr)
            else
            if  Catop(R) = 'X'  then
                bool := Parse(Rightop(R), 1, t, hdr)
            else
            if  Catop(R) = '*'  then
                begin hdr := hdl;
                bool := Parse(Rightop(R), 1, t, hdr)  end
            if  bool  then  h := hdr
            else
            begin
                hdl := hl;
                leftbool := Tryagain(Getnode(Leftop(R)), t, hdl)
            end
        end
    end
    Parse := if  bool  then true else  Parse(L, rpn+1, t, h)
end
```

The formal parameters of Parse have the meanings:

- L: a non-terminal symbol with or without a label representing a goal.
- rpn: a right-part alternative number for the production whose left part is L .
- t: the coordinates of the expected tail of the goal. t will always be defined.
- h: either the coordinates of the expected head of the goal or undefined (Λ) .

The auxiliary procedures called by Parse perform the following functions:

1. 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 .

2. Rightpart(x, n):

   Rightpart returns the n^{th} 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.

3. Prim(x):

   $x = y$ or y^{ℓ} . Prim returns true if y is of the form: p or (/p) or $(\sim(/p))$, where p is a labeled or unlabeled primitive class name.

4. Lookforprimitive(x, t, h):

   x is of the same form as the parameter of Prim.

   t is the expected tail location of x .

   h is the expected head location of x .

   Lookforprimitive calls a pattern recognition routine that attempts to find the primitive of x at the location defined by t and h . If h is undefined (Ω), t is the only constraint on the primitive; otherwise t and h must be satisfied by it. A successful search will return <u>true</u> and the head location (h) (if $h = \Omega$ at the call). 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 t and (possibly) the tail of h . Lookforprimitive returns <u>false</u> if all the constraints are not met.

5. 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>.

6. Catop(x), Leftop(x), Rightop(x):

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

7. Tryagain(goal,t,h), Getnode(x):

Tryagain is called when the left operand of a binary rightpart is successful but the right operand fails. Tryagain attempts to find a picture satisfying _another_ description generated by the left operand. It is implicitly assumed that at each call of Parse, a node is added to a parsing tree; this node or goal contains the parameters of Parse and is linked to its superior goal. The details of the tree construction appear in the listing of the implemented system given in the next section.

Getnode(x) retrieves the node constructed for the non-terminal symbol x .

Tryagain(goal,t,h) examines the branch of the tree starting at the node designated by goal and tries to find a parse of the picture which (a) satisfies the tail and head constraints, t and h, (b) has a description generated by one of the alternatives of the goal, and (c) is different from previous trials. Tryagain returns _true_ if successful and _false_ otherwise.

If Parse is successful, it returns _true_ and t and h will contain the coordinates of the tail and head of the picture; otherwise, Parse returns _false_. The first call of Parse is of the form: Parse(D,1,t,h), where D is the designated symbol of the grammar, t is the origin and h is undefined.

The flow in the left-to-right parsing algorithm is based on a case analysis of the elements of the standard form. The algorithm can be extended to include the construction of the parsing tree and natural semantics as is done in the implemented system below; then a successful parse of a picture α would yield its description $D(\alpha)$.

5.4 THE IMPLEMENTED PARSER

The primary purpose of the implementation was to investigate the benefits of this approach to picture processing by actually analyzing a non-trivial set of real pictures. For a number of reasons given in Chapter 7, a class of pictures produced in high energy particle physics was selected. These pictures - and all of the examples except the last of Chapter 4 - can be described in PDL using the operators $+$, $\times$, and $*$ alone.

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 $\{+, \times, *\}$ but is otherwise identical to PDL. Most of the discussion preceding the algorithm of the last section applies to the SPDL parsing system also.

5.4.1 GENERAL DISCUSSION OF THE SPDL PARSER

A schematic of the system organization is shown in Figure 5.2(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

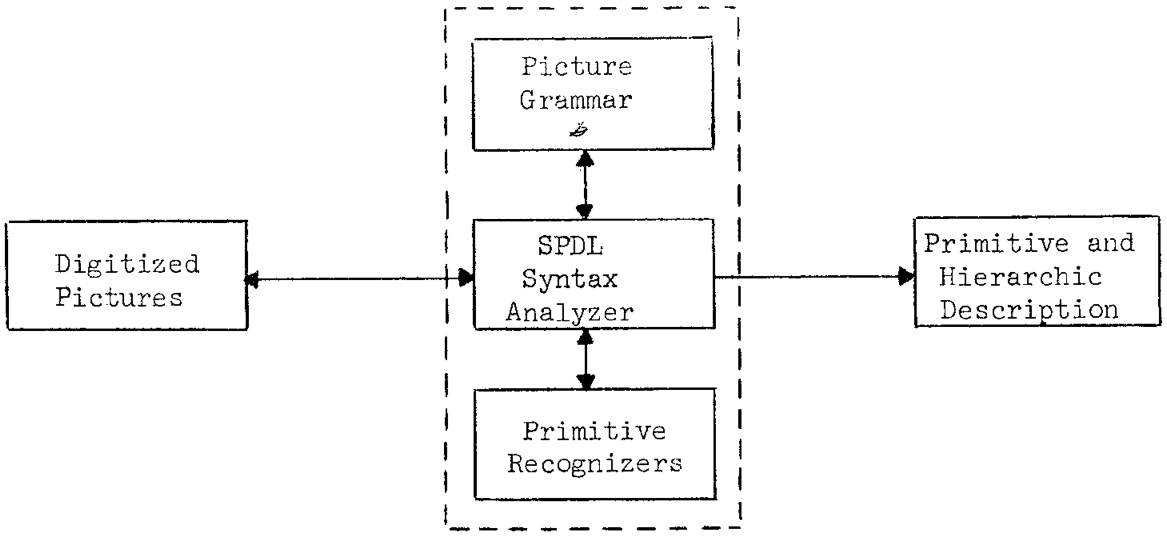


Figure 5.2(a) SPDL System Organization

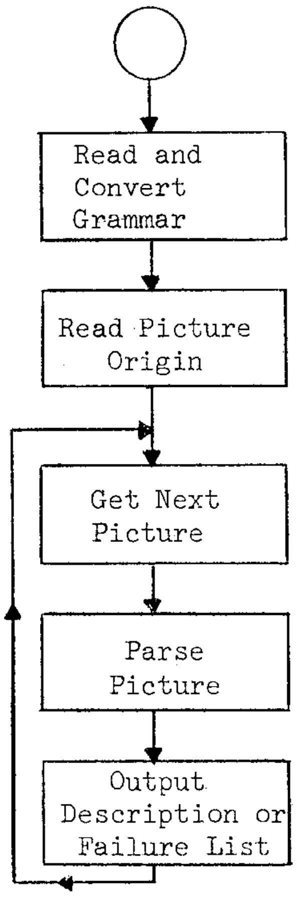


Figure 5.2(b) General Flow Chart of Program

Figure 5.2 The Implemented SPDL System

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 (IBM [1966], IBM [1967], IBM [1965]).

Figure 5.2(b) contains a general flow chart of the program. The input productions of $\mathcal{G}$ are of the form: $A \rightarrow (B\phi_b C)$ or $A \rightarrow D$, where A is a non-terminal symbol, D is a non-terminal symbol or primitive class name, $\phi_b \in \{+, \times, *\}$, and B and C may be <u>any</u> SPDL expression composed of primitive class names and non-terminal symbols; left-recursion is not provided for. Symbols must be less than 5 characters in length. Each production is first parsed (in the language sense) for well-formedness and then converted to standard form. Internally, the grammar is stored as a list structure to allow access to the definitions (right parts) of all non-terminal symbols. Each primitive class name has a pattern recognition routine number which is also read with the grammar. 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 (the details of the digitization process and the picture data structures used are discussed in Chapter 6); the input may be from cards or magnetic tape. The core of the system is the parser which is an iterative (non-recursive) version of the algorithm presented in section 5.3. Figure 5.3 contains the FORTRAN listing of the parse routine, PARSEP . Some of its features deserve mention. The parsing tree and goals are administered by means

```
                                    DATE  YS/360  FORTRAN H

          COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=50,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,MAP,NOEDIT,NOID

ISN 0002              SUBROUTINE PARSEP(X,Y,Z,PARSE)                                        00001000
ISN 0003              IMPLICIT INTEGER (A-Z)                                                00002000
ISN 0004              COMMON  /MGL/  MRGLIN(31)                                             00003000
ISN 0005             CCOMMON /GRAMC/ PRIM(100),PRSUB(100),NPRIM,PROD(100),PRODPT(100),      00004000
                     1    NPROD,GRAM(1000),NGRAM                                            00005000
ISN 0006             0COMMON /PSTACK/ GX(400),ALT(400),SUP(400),XSUP(400),LSUC(400),        00006000
                     1 RSUC(400),LOC(400),TAILPT(400),HEADPT(400),VALUE(3000),NV            00007000
ISN 0007              COMMON /BASYMB/ PLUS,TIMES,STAR,BLANK,LPAREN,RPAREN,DOLLAR,ORG        00008000
ISN 0008              COMMON /DEBUG/  DEBUGP                                                00009000
ISN 0009              COMMON /DG2250/ UBEAM,TEE,ACHE,SRT,EPM,ECF,EVM,TRU                    00010000
ISN 0010              COMMON /PICTUR/ PICT(10000),NP,YPT(1023),YCNT(1023),GT,GH             00011000
ISN 0011              COMMON /FAILR/ FPRIM(200),FVALPT(200),FBUFPT(200),NFPRIM              00012000
ISN 0012              COMMON /PT2250/ FPICT(1000),WPICT(1000),NFPICT,NWPICT,FT,FH,DELAY     00013000
ISN 0013              COMMON /SUBARG/ INDEX,XT,YT,ZT,XH,YH,ZH,XVALD(100),NXVALD,LC          00014000
ISN 0014              COMMON /WIND/ XW(500),YW(500),WBP(500),WCP(500),WIZE                  00015000
ISN 0015              COMMON /CELLS/ XC(200),YC(200),NCP(200),XCMIN(200),XCMAX(200),        00016000
                     1                YLAST(200),YCMIN(200),NC ,FSTW(200),LSTW(200)          00017000
ISN 0016              REAL XC,YC                                                            00018000
ISN 0017              COMMON /TRIPLE/ P1(200),P2(200),P3(200),THETA(200),CHAIN(200),NT      00019000
ISN 0018              REAL THETA                                                            00020000
ISN 0019              COMMON /LINES/ PF(31),PL(31),ANGLE(31),NPL(31),NL,PLINE(200)          00021000
ISN 0020              REAL PLINE,ANGLE                                                      00022000
ISN 0021              INTEGER X,Y,Z                                                         00023000
ISN 0022              LOGICAL PARSE                                                         00024000
                C****************************************************************** 00025000
                C     PERFORM A TOP-DOWN PARSE OF A PICTURE STARTING AT ORIGIN=(X,Y,Z).    00026000
                C     THE ALGORITHM IS A MODIFICATION OF THE CLASSICAL TOP-DOWN STRING     00027000
                C     PARSER.INSTEAD OF THE USUAL GETNEXTSYMBOL ROUTINE AT THE BOTTOM OF   00028000
                C     THE TREE,WE HAVE A PATTERN RECOGNITION PROGRAM ,LFP(GOAL,RESULT).    00029000
                C     LFP LOOKS FOR THE PRIMITIVE AT PARSINGSTACK(GOAL).IF FOUND,RESULT    00030000
                C     IS SET .TRUE.,OTHERWISE .FALSE..VALUE(I),I=1,NV STORES THE VALUES    00031000
                C     OF PRIMITIVES AS THEY ARE FOUND.GRAM(I) CONTAINS THE GRAMMAR         00032000
                C     DEFINING THE ACCEPTABLE PICTURE CLASS.                               00033000
                C     PARSING IS DONE VIA A STACK ,9 VARIABLES WIDE...                     00034000
                C              GX(G)=INDEX IN GRAM OF GOAL                                 00035000
                C              ALT(G)=ALTERNATIVE NO OF PRODUCTION RIGHT PARTS             00036000
                C              SUP(G)=INDEX IN STACK OF SUPERIOR GOAL                      00037000
                C              XSUP(G)=MT ,IF SUPERIOR GOAL IS A NON-TERMINAL SYMBOL       00038000
                C                     =LEFT,IF IT IS LEFT OPERAND OF A BINARY EXPRESSION   00039000
                C                     =RIGHT,IF THE RIGHT OPERAND                          00040000
                C                     =TOP IS G IS THE DESIGNATED SYMBOL                   00041000
                C              LSUC,RSUC(G)=LEFT AND RIGHT OPERAND SUCCESSOR STACK INDIC   00042000
                C              IF GOAL NOT BINARY EXP, ONLY LSUC(G) IS USED.               00043000
                C              LOC(G)=TAIL,HEAD,OR TLHD MEANING THAT TAILPT(G),HEADPT(G)   00044000
                C              ,OR BOTH POINT TO EXPECTED TAIL AND HEAD RESPECTIVELY       00045000
                C              IN VALUE().TAILPT,HEADPT=EXPECTED OR ACTUAL TAIL,HEAD INO   00046000
                C     THE FORMAT OF A VALUE LIST FOR A PRIMITIVE IS...                     00047000
                C          VALUE(I)=PRIMITIVE NAME                                         00048000
                C          VALUE(I+1)=LENGTH OF LIST ,.GE.6                                00049000
```

Figure 5.3

```
         C          VALUE(I+K),K=2,7=TAIL(X,Y,Z) AND HEAD(X,Y,Z).
         C          VALUE(I+K),K=8,VALUE(I+1)=REMAINING VALUES OF PRIMITIVE,IF ANY
         C       WHEN LFP IS SUCCESSFUL,ALT(G) WILL POINT TO INDEX IN VALUE().
         C          SETVAL(NAME,XT,YT,ZT,XH,YH,ZH,XVAL,NXVAL) STORES NAME,TAIL AND
         C          HEAD COORD,AND XVAL(I),I=1,NXVAL  IN VALUE() AND UPDATES INDEX
         C          STSTR(S,GX(S),ALT(S),SUP(S),XSUP(S),LOC(S),TAILPT(S),HEADPT(S)
         C               G,LSUC(G),RSUC(G))
         C          SETS STACK(S) TO TOP LINE,STACK(G) TO BOTTOM LINE ,AND G=S.
         C       IF  THE PARSE IS SUCCESSFUL  PARSE=.TRUE. AND THE PARSING STACK
         C       CONTAINS THE TREE., OTHERWISE,PARSE=.FALSE.
         C*****************************************************************
ISN 0023         LOGICAL RESULT
         C       SET PARAMETERS FOR XSUP(G) AND LOC(G).
ISN 0024            TOP=1
ISN 0025            AT=2
ISN 0026            RIGHT=3
ISN 0027            LEFT=4
ISN 0028            TAIL=1
ISN 0029            HEAD=2
ISN 0030            TLHD=3
         C       SET VALUE LIST FOR ORIGIN WITH COMMON TAIL,HEAD AT (X,Y,Z).
ISN 0031            NV=0
ISN 0032            CALL SETVAL(ORG,X,Y,Z,X,Y,Z,XVAL,0)
         C       INITIALIZE PARSING STACK TO FIRST PRODUCTION LIST IN GRAM().
         C       S=TOP OF PARSING STACK. G=INDEX OF CURRENT GOAL IN STACK.
ISN 0033            S=1
ISN 0034            G=1
ISN 0035            IF (GRAM(1).NE.1) GO TO 800
ISN 0037            CALL STSTR(S,1,1,0,TOP,TAIL,3,0,G,0,0)
         C*****************************************************************
         C          MAINLOOP...
         C       PROCEED DOWN THRU GRAMMAR SETTING UP GOALS IN THE PARSING STACK.
         C       EXIT TO SUCCESS OR FAILURE AFTER PRIMITIVE GOAL TESTED.
ISN 0038      11    I=GX(G)
ISN 0039            TYPE=GRAM(I)
         C       TYPE=1 FOR NT,=2 FOR PRIMITIVE,=3 FOR BINARY EXPRESSION.
ISN 0040      12    GO TO (100,300,200),TYPE
ISN 0041                 GO TO 800
         C       NT...
         C       CURRENT GOAL IS A NON-TERMINAL OR PRODUCTION.
ISN 0042     100         IF (ALT(G).GT.GRAM(I+2)) GO TO 500
ISN 0044                 S=S+1
ISN 0045                 IND=I+2+ALT(G)
ISN 0046        0        CALL STSTR(S,GRAM(IND),1,G,MT,LOC(G),TAILPT(G),
                1             HEADPT(G),G,S,0)
ISN 0047                 GO TO 11
         C       BIN...
         C       GOAL IS A BINARY EXPRESSION
ISN 0048     200         LOCP=TAIL
ISN 0049                 IF ((LOC(G).EQ.TLHD).AND.(GRAM(I+2).EQ.STAR)) LOCP=TLHD
```

Figure 5.3

```
                    S=S+1
      0             CALL STSTR(S,GRAM(I+1),I,G,LEFT,LOCP,TAILPT(G),
      1                  HEADPT(G),G,S,0)
                    GO TO 11
C       PRIMITIVE..
C       GOAL IS A PRIMITIVE.CALL RECOGNIZER AND EXIT TO SUCCESS(400)
C            OR FAILUREBACKUP(500).
  300               CONTINUE
C        DISPLAY RESIDUE PICTURE
               CALL DCRT(1)
                    CALL LEP(G,RESULT)
C        DISPLAY FOUND PICTURE
               CALL DCRT(2)
                    IF (RESULT) GO TO 400
                    GO TO 500
C*********************************************************************
C       SUCCESS...
C       BACK UP THE GOAL TREE UNTIL AN UNSATISFIED ONE (BINARY) IS FOUND.
C       SET THE CORRECT NODE VALUES ON THE WAY UP.THEN RETURN TO MAIN LOOP
  400          TYPE=XSUP(G)
  402          GO TO (410,420,430,440),TYPE
                    GO TO 850
C       TOP...
C       WE HAVE BACKED UP TO THE TOP.PARSE IS THUS SUCCESSFUL
  410               PARSE=.TRUE.
                   CALL PRSOUT(PARSE,S)
               RETURN
C       NT...
C       ADJUST HEAD,TAIL OF SUP(G) AND KEEP BACKTRACKING UP TREE.
  420               GD=SUP(G)
                    TAILPT(GD)=TAILPT(G)
                    HEADPT(GD)=HEADPT(G)
                    G=GD
                    GO TO 400
C       RIGHT... ADJUST HEAD AND BACKTRACK UP TREE.
  430               GD=SUP(G)
                    HEADPT(GD)=HEADPT(G)
                    G=GD
                    GO TO 400
C*********************************************************************
C       LEFT...
C       SET UP GOAL FOR RIGHT PART OF EXP. AND GO TO MAINLOOP.
  440               GD=SUP(G)
                    I=GX(GD)
                    S=S+1
                    CATOP=GRAM(I+2)
                    IF (CATOP.EQ.PLUS) GO TO 450
                    IF (CATOP.EQ.TIMES) GO TO 460
                    IF (CATOP.EQ.STAR)  GO TO 470
                         GO TO 800
```

Figure 5.3

```
C      PLUS...
  450              L=TAIL
                   IF (LOC(GD).EQ.TLHD) L=TLHD
    0              CALL STSTR(S,GRAM(I+3),1,GD,RIGHT,L,HEADPT(G),
    1                  HEADPT(GD),GD,LSUC(GD),S)
                   G=S
                   GO TO 11
C      TIMES...
  460              L=TAIL
                   IF (LOC(GD).EQ.TLHD) L=TLHD
    0              CALL STSTR(S,GRAM(I+3),1,GD,RIGHT,L,TAILPT(G),
    1                        HEADPT(GD),GD,LSUC(GD),S)
                   G=S
                   GO TO 11
C      STAR...
  4700             CALL STSTR(S,GRAM(I+3),1,GD,RIGHT,TLHD,TAILPT(G),
    1                        HEADPT(G),GD,LSUC(GD),S)
                   G=S
                   GO TO 11
C*********************************************************************
C      FAILUREBACKUP...
C      ENTER HERE WHEN A GOAL FAILS. BACK UP TREE LOOKING
C      FOR ANOTHER ALTERNATIVE TO TRY.
  500      TYPE=XSUP(S)
          IF (DEBUGP.EQ.1) GO TO 502
          CALL PRSOUT(.FALSE.,S)
  502      GO TO (510,520,530,540),TYPE
               GO TO 850
C      TOP...    WE HAVE REACHED TREE TOP. PARSE FAILS.
  510          PARSE=.FALSE.
          CALL PRSOUT(.FALSE.,S)
           RETURN
C      NT...     GET NEXT ALTERNATIVE AND GO TO MAIN LOOP.
  520          S=S-1
               G=S
               ALT(G)=ALT(G)+1
               GO TO 11
C      RIGHT... FOUND LEFT OPERAND IS IN S-1. KEEP BACKING UP.
  530          S=S-1
C      LEFT OPERAND MUST BE A FOUND PRIMITIVE. PUT ON FAILURE LIST.
              NFPRIM=NFPRIM+1
              I=GX(S)
              IF (GRAM(I).NE.2) GO TO 850
              FPRIM(NFPRIM)=GRAM(I+1)
              FVALPT(NFPRIM)=ALT(S)
C      REMOVE PRIMITIVE FROM FOUND BUFFER(FPICT).STORE IN RESIDUE (PICT).
               CALL SUBUF
               GO TO 500
C      LEFT ...  KEEP BACKING UP .
  540          S=S-1


               GO TO 500
C*********************************************************************
C      ERROR ROUTINES
  800      WRITE(6,900)
  900 FORMAT('0***PARSEP-GRAM(I) IS NOT CORRECT***')
           STOP
  850      WRITE(6,950)
  950 FORMAT            ('0***PARSEP-PROGRAM ERROR-STACK INCORRECT***')
           STOP
      END
```

Figure 5.3 The Parsing Subroutine

of a stack; the parsing tree and natural semantics are necessary data for the parse and are easily obtained from the stack. The part of the program that does the backtracking when a goal fails (FAILUREBACKUP in the listing) also stores the primitives that were found by this goal on a <u>failure list</u>.

When a goal is a terminal symbol, PARSEP calls the primitive recognition system LFP (Lookforprimitive). LFP first searches the failure list to see if the primitive was found previously. If not, the main picture recognition routine is called; the latter is illustrated in the simple examples of the next section. When a primitive is found, its value is stored in a list and a pointer to this list is placed in the stack; certain classes of primitives are eliminated from the picture after recognition.

On a successful parse, the program prints a stack representation of the parsing tree and natural semantics (the hierarchic description and primitive structural description), and the primitive value list (the primitive value description).

The 2250 cathode-ray tube display or "scope" 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.

5.4.2. SIMPLE EXAMPLES

Two simple examples illustrate the input and output data formats, the parsing stack and its evolution, and the form of the call to the picture pattern recognition routines.

Example 1: <u>Recognition of a Hand-Digitized A</u>

A plot of a hand-digitized "A" is shown in Figure 5.4(a). The digitizings were input from cards, "blown up" so they could be seen on the scope, and then analyzed by the SPDL system. Figure 5.4(b) contains the grammar for the picture class; the right parts may be punched in free format across a card. The primitives are STRT, DP, DM, and HP; DP, DM, and HP correspond to the primitives used in an earlier example of a PDL expression for an "A" (Figure 3.4). The Lookforprimitive routine (LFP) will call the primitive pattern recognition system (RECOGP) after searching the failure list unsuccessfully; RECOGP is listed in Figure 5.4(c). The user inserts the particular primitive recognizers, in this case, STRT01, DP02, DM03, and HP04, into RECOGP; the number (the index of the computed GOTO) and primitive name associated with each routine is input with the grammar. STRT01 finds the blank primitive STRT by simply retrieving the closest point to the picture origin (1,1,0) ("0" for the z coordinate indicates a 2-dimensional picture). The remaining routines all use the general line recognizer discussed in the next chapter; except for a small area around their tail and head, the points comprising a line are eliminated after recognition. The output after the successful parse of the "A" is listed in Figure 5.4(d). The primitive value description (PRIMITIVE VALUE LIST...) contains the name, the tail and head coordinates of each primitive ((XT,YT,ZT),(XH,YH,ZH)), and any other attributes - in this case none - that were returned by the recognition routine. The stack is interpreted from the top down (first element down) as follows:

1. S is the stack index.
2. If NAME[S] is a non-terminal symbol, line S represents the production named by it. The rightpart used by NAME[S] is Rightpart(NAME[S],ALT[S]), where Rightpart is defined in section 5.3. LSUC[S] points to the line of the stack defining the structure generated by NAME[S] .
3. If NAME[S] is a primitive class name, ALT[S] points to its value in the VALUE LIST (the indices in the VALUE LIST are not shown).
4. If NAME[S] is blank, then line S represents a production created by the program for the standard form. All of the created productions in SPDL are of the form: $(A\phi_b B)$, where A and B are created non-terminals. OP[S] contains ϕ_b . LSUC[S] and RSUC[S] point to the stack entries for the left and right operands respectively.
5. The comments at the beginning of the listing of PARSEP (Figure 5.3) explain the remainder of the entries.

The tree of Figure 5.4(e) is easily constructed from the parsing stack.

```
WINDOW PLOT....XL,YL,XH,YH=      1     950      60    1023
```

```
                      * *
                    * * * *
                    *     *
                  * *     * *
                  *         *
                *             *
              * *             * *
              *                 *
            * *                 * *
            * * * * * * * * * * * *
          *                         *
        * *                         * *
        *                             *
      * *                             * *
      *                                 *
    * *                                   *
  * *                                     * *
  *                                         *
* *                                         * *
```

Figure 5.4(a) Plot of Hand-Digitized "A"

```
AA = (STRT + A)

A  = ((DP + ((DP + DM) * HP)) + DM)
```

Figure 5.4(b) Syntax for A's

```
                              DATE   YS/360   FORTRAN H

      COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

                 SUBROUTINE RECOGP(RESULT)
                 IMPLICIT INTEGER (A-Z)
                 COMMON  /MGI/  ARGLIN(31)
                 COMMON /GRAMC/ PRIM(100),PRSUB(100),NPRIM,PROD(100),PRODPT(100),
                1     GPRSUB,GRAM(1000),NGRAM
                 COMMON /PSTACK/ GX(400),ALT(400),SUP(400),XSUP(400),LSUC(400),
                1 RSUC(400),LOC(400),TAILPT(400),HEADPT(400),VALUE(3000),NV
                 COMMON /PICTUR/ PICT(10000),NP,YPT(1023),YCNT(1023),GT,GH
                 COMMON /SUBARG/ INDEX,XT,YT,ZT,XH,YH,ZH,XVALD(100),NXVALD,LC
                 COMMON /WIND/ XW(500),YW(500),WBP(500),WCP(500),WIZE
                 COMMON /CELLS/ XC(200),YC(200),NCP(200),XCMIN(200),XCMAX(200),
                1                YLAST(200),YCMIN(200),NC ,FSTW(200),LSTW(200)
                 REAL XC,YC
                 COMMON /TRIPLE/ P1(200),P2(200),P3(200),THETA(200),CHAIN(200),NT
                 REAL THETA
                 COMMON /LINES/ PF(31),PL(31),ANGLE(31),NPL(31),NL,PLINE(200)
                 REAL PLINE,ANGLE
                 LOGICAL RESULT
          C      ****************************************************************
          C      CALL THE PRIMITIVE RECOGNITION ROUTINE ,PRSUB(INDEX), FOR THE
          C      PRIMITIVE PRIM(INDEX).LC=1,2,3 CONSTRAINS THE PRIMITIVE COORDS TO
          C      TAIL(XT,YT,ZT),HEAD(XH,YH,ZH),OR BOTH. IF THE RECOGNITION ROUTINE
          C      IS SUCCESSFUL,RESULT=.TRUE., (XT,YT,ZT) AND (XH,YH,ZH) ARE SET TO
          C      THE TAIL AND HEAD OF THE PRIMITIVE,XVALD(I),I=1,NXVALD CONTAIN THE
          C      REMAINING (IF ANY) VALUES. OTHERWISE,RESULT=.FALSE.
          C      THE RECOGNITION ROUTINES PROVIDE FOR 'SLOP' AROUND THE REQUIRED
          C      TAIL AND/OR HEAD.
          C      THE RECOGNITION ROUTINES WILL ELIMINATE THE FOUND PRIMITIVE FROM
          C      THE RESIDUAL PICTURE.
          C      ****************************************************************
                   SUB=PRSUB(INDEX)
          C      THIS SECTION MUST BE INSERTED BY THE USER.
          C      ****************************************************************
          C      THESE ARE RECOGNIZERS FOR AN 'A'.
                     GO TO (1,2,3,4),SUB
                     GO TO 800
                1           CALL STRT01(RESULT)
                                 IF (RESULT) GO TO 500
                                 RETURN
                2           CALL DP02(RESULT)
                                 IF (RESULT) GO TO 500
                                 RETURN
                3           CALL DM03(RESULT)
                                 IF (RESULT) GO TO 500
                                 RETURN
                4           CALL HP04(RESULT)
                                 IF (RESULT) GO TO 500
                                 RETURN
          C      SEARCH IS SUCCESSFUL UPDATE VALUE LIST.


              500       CALL SETVAL(INDEX,XT,YT,ZT,XH,YH,ZH,XVALD,NXVALD)
                        RETURN
          C       INDEXING ERROR
              800       WRITE(6,900)
              900 FORMAT('0**RECOGP--INDEXING ERROR.SUBR DOES NOT EXIT.**')
                        STOP
                  END
```

Figure 5.4(c) The Primitive Pattern Recognition System

```
PICTURE PARSE WAS SUCCESSFUL..
 NATURAL SEMANTICS OF PICTURE
 PARSING TREE...
    S   GX  ALT   SUP XSUP LSUC RSUC   LOC TLPT HDPT          NAME  OP

    1    1    1    0    1    2    0     1    3   54          AA
    2    5    1    1    2    3    4     1    3   54                 +
    3  806    9    2    4**********     1   11   14          STRT
    4    9    1    2    3    5    0     1   14   54          A
    5   25    1    4    2    6   13     1   14   54                 +
    6   21    1    5    4    7    8     1   14   46                 +
    7  802   17    6    4*****59312     1   19   22          DP
    8   17    1    6    3    9   12     1   22   46                 *
    9   13    1    8    4   10   11     1   22   38                 +
   10  802   25    9    4**********     1   27   30          DP
   11  800   SS    9    S**********     1   35   38          DM
   12  804   41    8    3**********     3   43   46          HP
   13  800   49    5    3**********     1   51   54          DM
 PRIMITIVE VALUE LIST...
NAME    XT   YT   ZT   XH   YH   ZH   OTHER VALUES

STRT     1    1    0    6  932    0
DP       6  932    0   28  970    0
DP      28  970    0   52 1002    0
DM      52 1002    0   72  970    0
HP      28  970    0   72  970    0
DM      72  970    0   96  930    0
```

Figure 5.4(d) Final Output After a Successful Parse

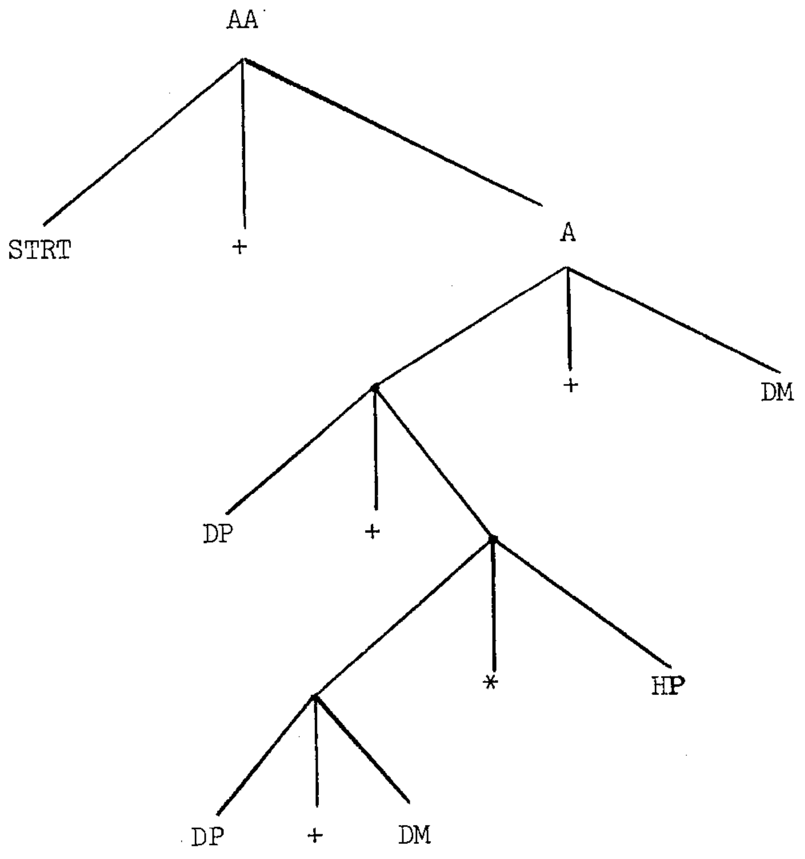


Figure 5.4(e) Parsing Tree Graph

Figure 5.4 Parsing a Digitized "A"

Example 2: Recursion and Backtracking

This example was run on an earlier version of the system that accepted only primitive value lists as the picture representation; primitive recognition is done by searching these lists. The recursive grammar of Figure 5.5(a) generates the SPDL expressions $\{\sum_{i=1}^{n} (DP + DM) | n \geq 1\}$, where $\sum_{i=1}^{n} a = (a + (a + (a + (a + \ldots + (a + a))) \ldots))$. DP and DM, and the sawtooth input picture are illustrated in Figure 5.5(b); the PICTURE VALUE LIST contains the name, and tail and head coordinates of each primitive. The goal $S \rightarrow ((DP + DM) + S)$ (line 11 in the stack of Figure 5.5(c)) fails on the last S (line 16); the parser then backtracks to line 11 (SUP[SUP[16]]) and sets up the goal for the second alternative of S: $S \rightarrow (DP + DM)$. This goal is successful and the final picture description is printed (Figure 5.5(d)).

$$S = ((DP + DM) + S)$$
$$S = (DP + DM)$$

Figure 5.5(a) Sawtooth Syntax $(\sum_{i=1}^{n} (DP + DM))$

PICTURE VALUE LIST

	XT	YT	ZT	XH	YH	ZH
ORG	1	1	0	1	1	0
DP	1	1	0	5	5	0
DP	9	1	0	13	5	0
DP	17	1	0	21	5	0
DM	5	5	0	9	1	0
DM	13	5	0	17	1	0
DM	21	5	0	25	1	0
$						

Figure 5.5 continued on next page.

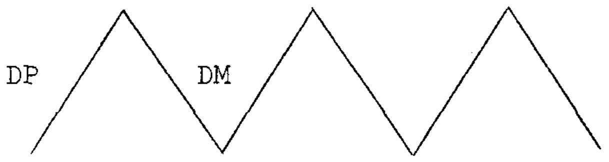


Figure 5.5(b) Input Picture n=3

```
PARSING TREE...

  S   GX  ALT   SUP XSUP LSUC RSUC  LOC TLPT HDPT          NAME  OP

  1    1    1    0    1    2    0    1    3    0          S
  2   10    1    1    2    3    6    1    3    0                 +
  3    6    1    2    4    4    5    1    3   22                 +
  4  802    9    3    4    5    6    1   11   14          DP
  5  800   17    3    3    6    7    1   19   22          DM
  6    1    1    2    3    7    0    1   22    0          S
  7   10    1    6    2    8   11    1   22    0                 +
  8    6    1    7    4    9   10    1   22   38                 +
  9  802   25    8    4   10    0    1   27   30          DP
 10  800   33    8    3    0*****    1   35   38          DM
 11    1    1    7    3   12    0    1   38    0          S
 12   10    1   11    2   13   16    1   38    0                 +
 13    6    1   12    4   14   15    1   38   54                 +
 14  802   41   13    4    0*****    1   43   46          DP
 15  800   49   13    3    0*****    1   51   54          DM
 16    1    3   12    3   17    0    1   54    0          S
```

Figure 5.5(c) Last Goal (S = 16) Leads to Failure

```
PICTURE PARSE WAS SUCCESSFUL..
 NATURAL SEMANTICS OF PICTURE
 PARSING TREE...
    S   GX  ALT  SUP XSUP LSUC RSUC  LOC TLPT HDPT         NAME  OP

    1    1    1    0    1    2    0    1    3   54         S
    2   10    1    1    2    3    6    1    3   54               +
    3    6    1    2    4    4    5    1    3   22               +
    4  802    9    3    4    5    6    1   11   14         DP
    5  800   17    3    3    6    7    1   19   22         DM
    6    1    1    2    3    7    0    1   22   54         S
    7   10    1    6    2    8   11    1   22   54               +
    8    6    1    7    4    9   10    1   22   38               +
    9  802   25    8    4   10    0    1   27   30         DP
   10  800   33    8    3    0*****    1   35   38         DM
   11    1    2    7    3   12    0    1   38   54         S
   12   14    1   11    2   13   14    1   38   54               +
   13  802   41   12    4   14   15    1   43   46         DP
   14  800   49   12    3    0*****    1   51   54         DM
 PRIMITIVE VALUE LIST...
NAME    XT   YT   ZT   XH   YH   ZH   OTHER VALUES

DP       1    1    0    5    5    0
DM       5    5    0    9    1    0
DP       9    1    0   13    5    0
DM      13    5    0   17    1    0
DP      17    1    0   21    5    0
DM      21    5    0   25    1    0
```

Figure 5.5(d) Final Picture Description

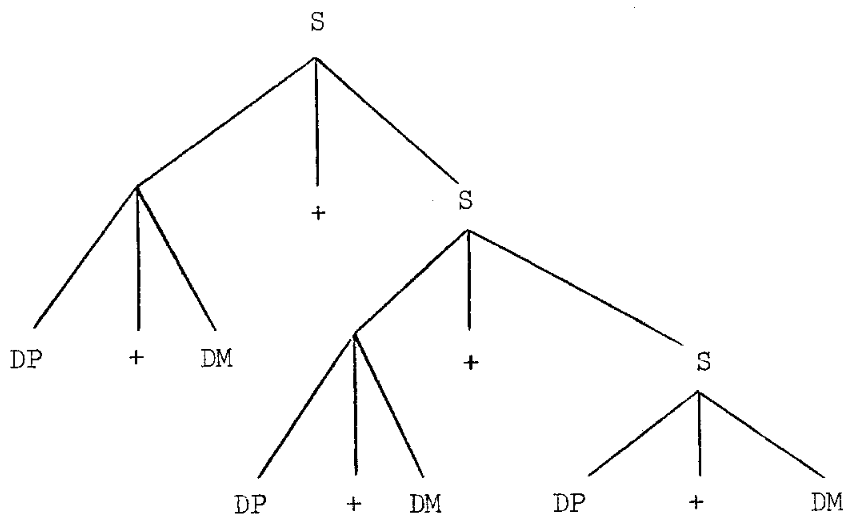


Figure 5.5(e) Graph of Parsing Tree

Figure 5.5 Sawtooth Parse

CHAPTER 6

PRIMITIVE RECOGNITION

This chapter describes the methods developed or modified by the author for the recognition of line segments, "blob" patterns, and some types of blank and don't care primitives. Most pictures contain line-like elements and efficient general techniques for their recognition are still lacking. Blobs are small connected sets of digitizings that are either meaningful in their own right or may be combined to form primitives.

Both the characteristics of the recognizers and the computer data structures depend to a limited extent on the picture digitization process. The input pictures are assumed to be digitized by a device similar to the Hummingbird machines.

6.1 HUMMINGBIRD AUTOMATIC FILM DIGITIZERS

"Hummingbird" is the name given to two spark chamber film digitizers developed at the Stanford Linear Accelerator Center by J. Van der Lans (Van der Lans [1967], Miller and Van der Lans [1967]). These machines are high-precision flying spot scanners employing the electron beam in a cathode ray tube (CRT) as a spot generator. The spots are deflected across the face of the CRT in a TV-type raster scan mode. A lens system images the spots on the film (35 mm. or 70 mm.). Behind the film is a photomultiplier, which senses the amount of light passing through the film at each spot position; the firing threshold of the photomultiplier can be adjusted over 16 signal levels. The spot on the CRT sweeps across Y scan lines with X least counts (identifiable spot positions) per line, where Y may be 2^9, 2^{10}, 2^{11}, or 2^{12}, and X is 2^{12} or 2^{14}, depending on the particular Hummingbird. The spot coordinates can be defined as the pair (x, y) = (least count, scan line number); x and y counters are synchronized with the spot sweep to allow the output of the spot positions. Both digitizers are presently connected to the IBM 360 model 75 computer, but were earlier attached to a model 50. Orders from the computer control the entire digitization process, including moving of film and selected digitizing of windows (small rectangles) in a frame.

The devices, in their normal mode of operation, return only the center coordinates of each dark area across a scan line that produces a photomultiplier signal below a given threshold. This results in a significant data reduction but biases the digitizings: line-like dark areas that are oriented in the y direction are very accurately digitized; similar areas in the x-direction are poorly digitized and often bear little resemblance to the actual picture. This is not a serious problem for particle physics film since tracks are usually oriented in one direction. For other types of patterns, this bias can make a picture unrecognizable; experience with real film has indicated that, in many of these cases, a multi-level rather than a binary digitization is necessary for recognition.

A complex and sophisticated programming system written by C. Dickens (Dickens [1967a]) allows a user to control and view the digitization process via the 2250 display, and associated light-pen and typewriter. This includes:

1. viewing the digitized picture on the scope,

2. viewing enlargements of windows of the pictures and varying the magnification factor,
3. selection of the particular Hummingbird,
4. setting of the photomultiplier threshold,
5. selection of the scan line density and least count,
6. specification of rectangular subareas of a frame for digitization,
7. adjustment of film and moving from frame to frame,
8. commands for the actual digitization, and
9. writing of the digitizations on magnetic tape or the line printer.

The Hummingbird output consists of the string of coordinates:

$$0 y_1 x_{11} x_{12} \ldots x_{1n_1} 0 y_2 x_{21} x_{22} \ldots x_{2n_2} 0 \ldots\ldots 0 y_m x_{m1} x_{m2} \ldots x_{mn_m}$$

where y_i is the scan line number and $x_{i1} x_{i2} \ldots x_{in_i}$ are the locations of the "hits" on scan line y_i. The points are ordered by x within y. This leads to the picture data structure used by the primitive recognizers in the SPDL system:

A digitization of 1024 scan lines with 4096 counts per line was assumed.

Y[i] points to the first hit for scan line i; Y[i] = 0 if no hits occurred.

Xcount[i] = number of hits on scan line i, i = 1, 1024 .

Then Xcoord[Y[i]] to Xcoord[Y[i] + Xcount[i]-1] contains the ordered list of x coordinates for the hits on scan line i .

The above structure was very convenient for accessing small windows during primitive recognition.

6.2 RECOGNITION OF BLOBS

In many applications, blobs of various types are picture primitives. Examples are: sparks appearing in particle physics spark chamber film; blobs representing "bits" in a data box; and characters, such as letters, digits, and punctuation. The tail and head are usually chosen at the blob center. The formation of blobs is also a useful first step--equivalent to preprocessing--for the recognition of more complex primitives.

The method for finding blobs is a simplified version of the cell construction algorithm of Clark and Miller [1966]. Given an ordered set of coordinates $\{P_i = (x_i, y_i) | (y_i \leq y_j) \wedge (y_i = y_j \supset x_i < x_j),\ i < j,\ i, j = 1, n\}$ representing hits in a picture window, and values for the parameters δx, δy, dy, and n_{min}, the blob recognizer groups ordered subsets of these points into cells (blobs) such that for each cell $C = \{P_{c_1}, P_{c_2}, \ldots, P_{c_m}\}$,

1. $|x_{c_i} - x_{c_j}| \leq \delta x$ and $|y_{c_i} - y_{c_j}| < \delta y$,
2. $|y_{c_i} - y_{c_{i+1}}| \leq dy$,
3. $|C| \geq n_{min}$, where $|C|$ is the number of points in cell C, and
4. $|C|$ is (almost) maximum.

Since the width of the dark areas in the x-direction is not known (see last section), a dx parameter corresponding to dy was not included. The output of the routine is an ordered list of the centers of gravity of each cell (ordered in the same manner as the input). Pointers from cells to their contained points and from points to their cells, if any, are also computed.

The cell into which a point is placed is sometimes dependent on the order of examination of the points and the cells. While this is not desirable in general, the differences are insignificant compared to errors introduced by the digitization process, photography, and other "noise" producers.

6.3 A GENERAL PURPOSE LINE RECOGNIZER (GPLR)

6.3.1 THE LINE RECOGNITION PROBLEM

A straight line in the plane can be defined abstractly as the set of points (x, y) satisfying the linear algebraic equation $ax + by + c = 0$, where a, b, and c are constants (real numbers), and $|a|+|b| \neq 0$; this line has no width, occupies no area, and has no irregularities. One cannot find such a clean definition for the class of elements that a human might call a line in a picture--indeed, what is interpreted as line-like depends to a great extent on the "eyes of the beholder". Like most general concepts, a precise definition is elusive; one can find patterns that are not called lines in many cases, yet would satisfy particular attempts at line definition. A trivial example is two points in a picture; they define a line exactly, but would rarely be interpreted in this way.

Examples of patterns that might be usefully classified as lines are shown in Figure 6.1. Qualitatively, a rectangle of small width to length ration can be placed over each line; the line roughly "fills" the rectangle along its length. The points on each line satisfy a linear algebraic equation to within a given tolerance according to some criterion of goodness of fit. Irregularities and difficulties that must be taken into account in a general system include:

1. local non-linearity,
2. local linearity but in a different orientation than the line,
3. appearance in a field containing other patterns that may intersect the line,
4. variable width, and
5. many gaps along the line length.

GPLR is designed to cope with these problems and recognize the variety of line-like elements of Figure 6.1.

There have been a large number of techniques developed in the past for line recognition. Most of these apply only to limited classes and/or are often very complex logically. Local preprocessing is frequently employed to reduce some of the irregularities mentioned above (Dinneen [1955], Unger [1959], Narasimhan [1964]). The most common method for recognition of short, fixed-length line segments is simple template matching (Bomba [1959], Roberts [1963], Rosen and Nilsson [1966]). Hardware or software line masks of fixed size and orientation are passed over the picture; the classification decisions are based on correlations with the masks. While this has proved successful for some pictures, it cannot, for example, produce useful results for lines consisting of non-linear blobs with many gaps unless the template size is large. The restriction to a fixed template size means that the digitization accuracy must be compatible with the properties of the line; in particular when the digitization is too fine for a line, the line segments produced by fixed-size templates might be meaningless. A variable size line template built into hardware in the PEPR system of Pless and Rosenson (Pless et. al. [1965]) for bubble chamber photograph analysis overcomes this difficulty; the

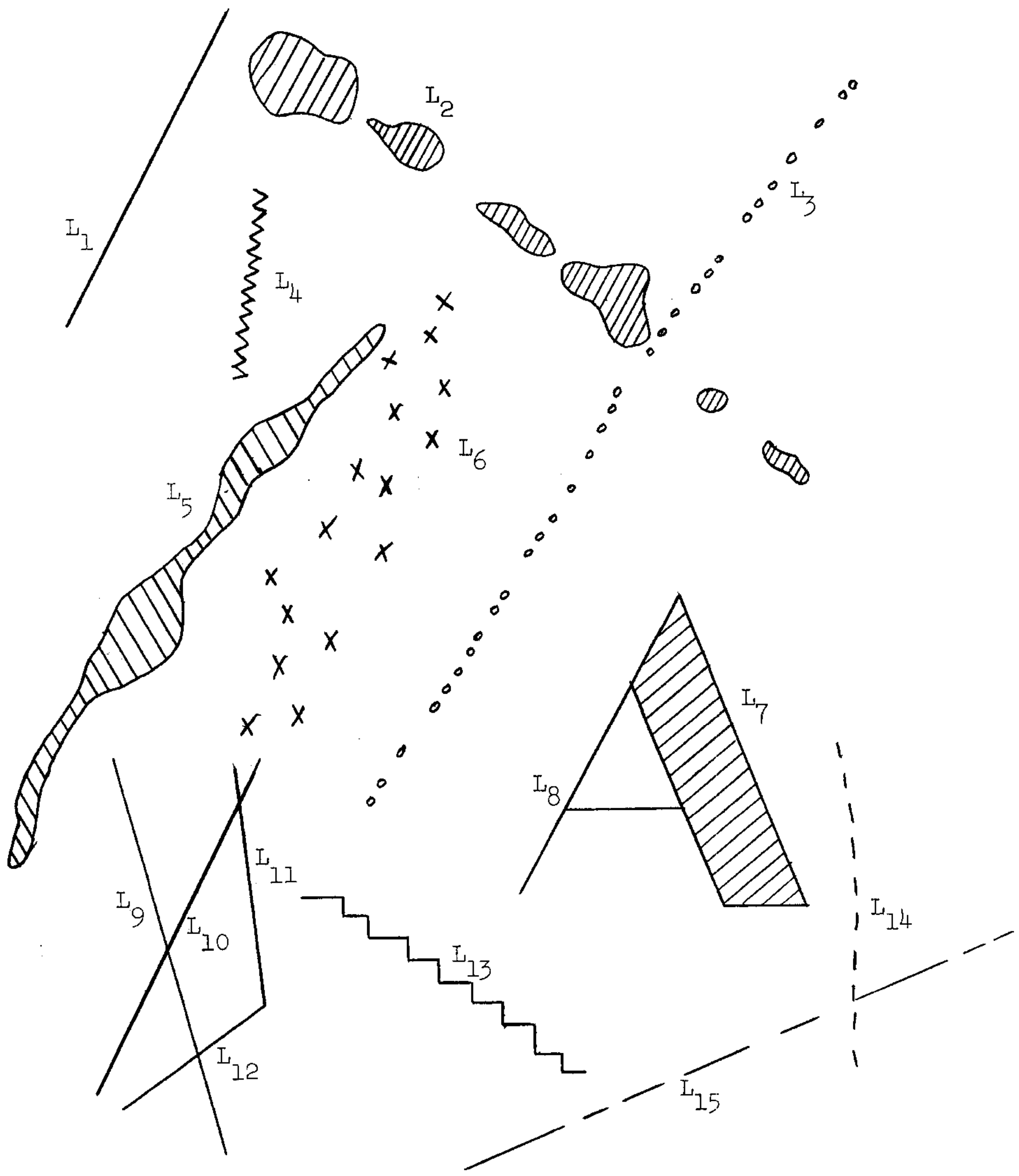


Figure 6.1 Examples of Line-Like Patterns

problem then becomes one of linking roughly collinear line segments together. Another set of techniques that is frequently employed either in direct recognition or for linking segments utilizes linear least squares methods (Roberts [1963], Miller [1966], McGee [1966]). These methods produce a good quantitative estimate of the linearity of the data, but usually require that the line be isolated. When lines are overlapped with other patterns, the method has to be modified considerably to keep on the "right track". Finally, several interesting line following or linking techniques have been developed; these are usually based on successive fitting and extrapolation along a line. The systems for track recognition in particle physics pictures are generally of this last type (Clark and Miller [1966], Marr and Rabinowitz [1966], Moorhead and Powell [1965]). This brief discussion is far from exhaustive, but covers most of the major approaches to line recognition.

6.3.2 GPLR

The methods used in GPLR are a major extension and modification of those employed by Marill et. al. [1963] for line segment formation in their CYCLOPS-1 system. Given a set of points representing digitizings, their system works as follows:

> The formation of line segments proceeds in three stages: triplet formation, triplet chaining and segment connecting. To form a triplet, we select a point and try to find two nearby points which are on opposite sides of, and approximately co-linear with, the original point. The chaining process begins after all possible triplets have been formed. Two triplets are joined provided they have two common points in the proper order. For example, the triplets abc and bcd will form the segment abcd . This segment may be extended by combining it with other triplets according to the same rule. When the current segment cannot be extended any further,

the process is repeated with the remaining triplets. Chaining terminates when no segment can be extended further. (Marill et. al. [1963]).

Two segments are linked if they are close together, have similar curvatures and slopes about the vicinity of their endpoints, and can be extrapolated to meet each other.

GPLR has a similar flow. It can be divided into four phases:

1. preprocessing by blobbing,

2. formation of collinear triples,

3. linking of triples into chains, and

4. merging chains into lines.

The digitized points are first blobbed by the routine described in Section 6.1; this process not only reduces the data, smooths local irregularities, and eliminates some noise, but it deliberately creates gaps in lines. The appearance of gaps in line-like elements has been a vexing problem; here, they will often assist the recognition. The parameters δx, δy, dy and n_{min} are selected experimentally and reflect some of the characteristics of the lines, the pictures in which they are embedded, and the digitization process. The remainder of the phases are almost independent of the digitization method. The output of the blobbing phase is the set of ordered cell centers $\{P_i = (x_i, y_i) | i = 1, n\}$.

The collinear triplet formation routine examines each point P_i, $i = 1, n-2$ in turn and builds all triples $T = (P_i, P_j, P_k)$, $i < j < k$, such that

1. $|x_i - x_j| \leq \Delta x \wedge |x_i - x_k| \leq \Delta x \wedge |x_j - x_k| \leq \Delta x$,

2. $|y_i - y_k| \leq \Delta y$, and

3. $\frac{h_j}{|P_iP_k|} \leq \epsilon$, where h_j is the length of the perpendicular from P_j to the line segment defined by P_i and P_k, and $|P_iP_k|$ is the length of that segment.

Δx, Δy, and ϵ are input parameters. Δx and Δy are functions of the expected inter-cell distances and gap size; they restrict the points under consideration to a small rectangle above P_i . It is generally advisable to choose Δx and Δy so that there is more than one triple emanating from a point P_i on a line; thus, one might have the triples: (P_i, P_{j_1}, P_{k_1}), (P_i, P_{j_1}, P_{k_2}), (P_i, P_{j_2}, P_{k_1}), etc., all of which are on the same line. This built-in redundancy is valuable for recognition through locally confused regions and where the local linearity of points is not a constant function of the segment length. ϵ is a co-linearity test parameter and represents the allowable upper bound on the width to length ration r of the minimum rectangle enclosing (P_i, P_j, P_k) . r is computed by taking advantage of the following equalities:

$$r = \frac{h_j}{|P_iP_k|} = \frac{2 \times (\frac{1}{2}|P_iP_k|h_j)}{|P_iP_k|^2}$$

$$= \frac{2 \times (\text{Area of Triangle with vertices } P_i,\ P_j, \text{ and } P_k)}{|P_iP_j|^2}$$

$$= \frac{|\Delta x_{ij}(y_i+y_j) - \Delta x_{kj}(y_k+y_j) - \Delta x_{ik}(y_i+y_k)|}{\Delta x_{ik}^2 + \Delta y_{ik}^2}$$

where $\Delta x_{ij} = x_j - x_i$. The last formula is used by the program. There is little in the literature on testing for collinearity of three points; the above test derived by the author is simpler than one based on least squares, yet reflects the intuitive notion of collinearity. The collinearity test implicitly assumes that the line formed by a successful triple is oriented from P_i to P_j; the only instances when this might not be true would be for some almost horizontal triples; the linking phase provides for this possibility. This method for recognizing small line segments (collinear triplets) is analogous to a <u>variable-size</u> template matching process. When a triple passes the collinearity test, an approximation to the angle that P_i and P_k makes with the x-axis is also computed.

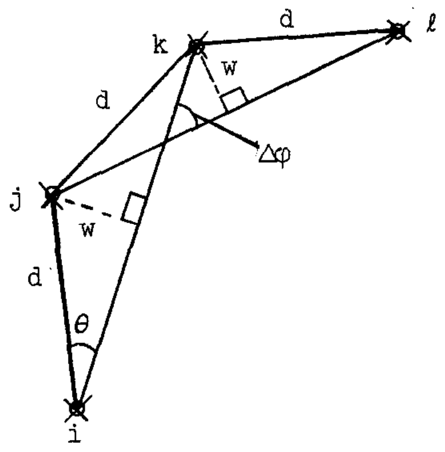
The next phase links triples into collinear chains. Two triples $T_i = (P_{i_1}, P_{i_2}, P_{i_3})$ and $T_j = (P_{j_1}, P_{j_2}, P_{j_3})$ are linked if $((i_2 = j_1) \wedge (i_3 = j_2))$. Consequently, a chain consisting of n triples will contain $n+2$ unique points. An approximate angle is computed for each chain by averaging the angles of its triples. Chains are merged into lines in the final step. Two chains are merged if they have at least one point in common and their angles are within $\Delta\theta$ of each other; $\Delta\theta$ is an input parameter.

A complete list of the parameters of GPLR is: $(\delta x, \delta y, dy, n_{min}, \Delta x, \Delta y, \epsilon, \Delta\theta)$. The output consists of the number of lines found, the angle of each line, the number of cells on each line, and a pointer list for each cell indicating which lines, if any, it is a part of. The lines recognized by GPLR may intersect each other; there is an additional routine that computes these intersection points for a selected line. Lines may be selected by specifying a length, angle, and location tolerance.

GPLR has several advantages compared to other line recognition schemes:

1. It is applicable to a wide variety of line-like elements (Figure 6.1).
2. Pattern fields containing intersecting lines can be classified.
3. It is simple and relatively efficient.

There are, however, a number of pictures and applications for which GPLR is not adequate. Physicists divide particle track analysis into two distinct phases--scanning and measuring. Scanning corresponds to basic pattern classification--does a track pattern of a given class or set of classes exist in a picture, and if so, roughly where? The measurement phase is concerned with the accurate computation of properties of the previously recognized or scanned patterns. In terms of this dichotomy, GPLR is a scanning or recognition system; more complex techniques are required for accurate measurements. One can envision pictures where the output of GPLR would have almost no meaning unless accompanied with some auxiliary processing; for example, a window that is dense in random digitizings would yield a large number of lines oriented in many directions. If the co-linearity parameter, ϵ, is not carefully selected, GPLR can classify curves and even nonsense as lines. The following simplified analysis of a "worst case" illustrates this point:



Assume that the points i, j, k, and ℓ are located such that

$$|ij| = |jk| = |k\ell| = d,$$

and that the triples (i, j, k) and (j, k, ℓ) just pass the collinearity test so that

$$\epsilon = \frac{w}{2\sqrt{d^2-w^2}} \approx \frac{w}{2d} \qquad \text{for} \quad w \ll d \ .$$

Then $\angle\underline{jik} = \theta = \angle\underline{jki} = \angle\underline{kj\ell}$. The two triples will be chained and the difference between the angles of the lines formed by their end points is $\Delta\varphi = 2\theta = 2 \sin^{-1}(\frac{w}{d}) \approx 2 \sin^{-1}(2\epsilon)$. If this "worst case" occurred for n+1 triples, there would be a total angle change of $2n \sin^{-1}(\frac{w}{d})$ $\approx 2n \sin^{-1}(2\epsilon)$. For large n, this could be disastrous; on the other hand, curved segments could be recognized in this manner if additional logic were inserted in the chaining routine to ensure that Δφ does not change sign.

With the proper choice of parameters, GPLR has proved to be extremely effective for recognition within the SPDL system. Here the system is directed to look for a member of a given class of lines in a specified area of the picture; if lines are concatenated together, a "slop" parameter is used as a tolerance around the intersection points. The simple digitized "A" of the last chapter was recognized in this way; Chapter 7 employs GPLR within SPDL in a complex setting.

The next section describes a non-trivial application of GPLR that illustrates many of its strengths and weaknesses.

6.3.3 RECOGNITION OF BUBBLE CHAMBER TRACKS BY GPLR

Figure 6.2(a) is a plot of a set of digitizings from a bubble chamber picture; the set is a modification of those that appeared in a picture in Narasimhan's paper [1964]. The latter picture was re-digitized by hand, simulating the Hummingbird. This picture is a particularly good test of GPLR. It contains many lines, some of which are overlapping; the lines vary in length, thickness, and point density; there are many gaps in the lines and some of the picture elements may or may not be lines depending on the interpretation. There are 304 digitizings in the picture.

The picture was input to GPLR. The results of the blobbing are shown in Figure 6.2(b). Parameters used were $\delta x = 1$, $\delta y = 3$, $dy = 2$, and $n_{min} = 1$. The cells are numbered from the bottom of the figure from 01 to 99 and then starting from 01 again near the top (meaning 101); the low order digit of the cell number lies on the raster unit nearest to the (floating point) cell center. 114 cells were constructed. As can be seen by comparing Figures 6.2(a) and 6.2(b), the essential details of the picture are retained.

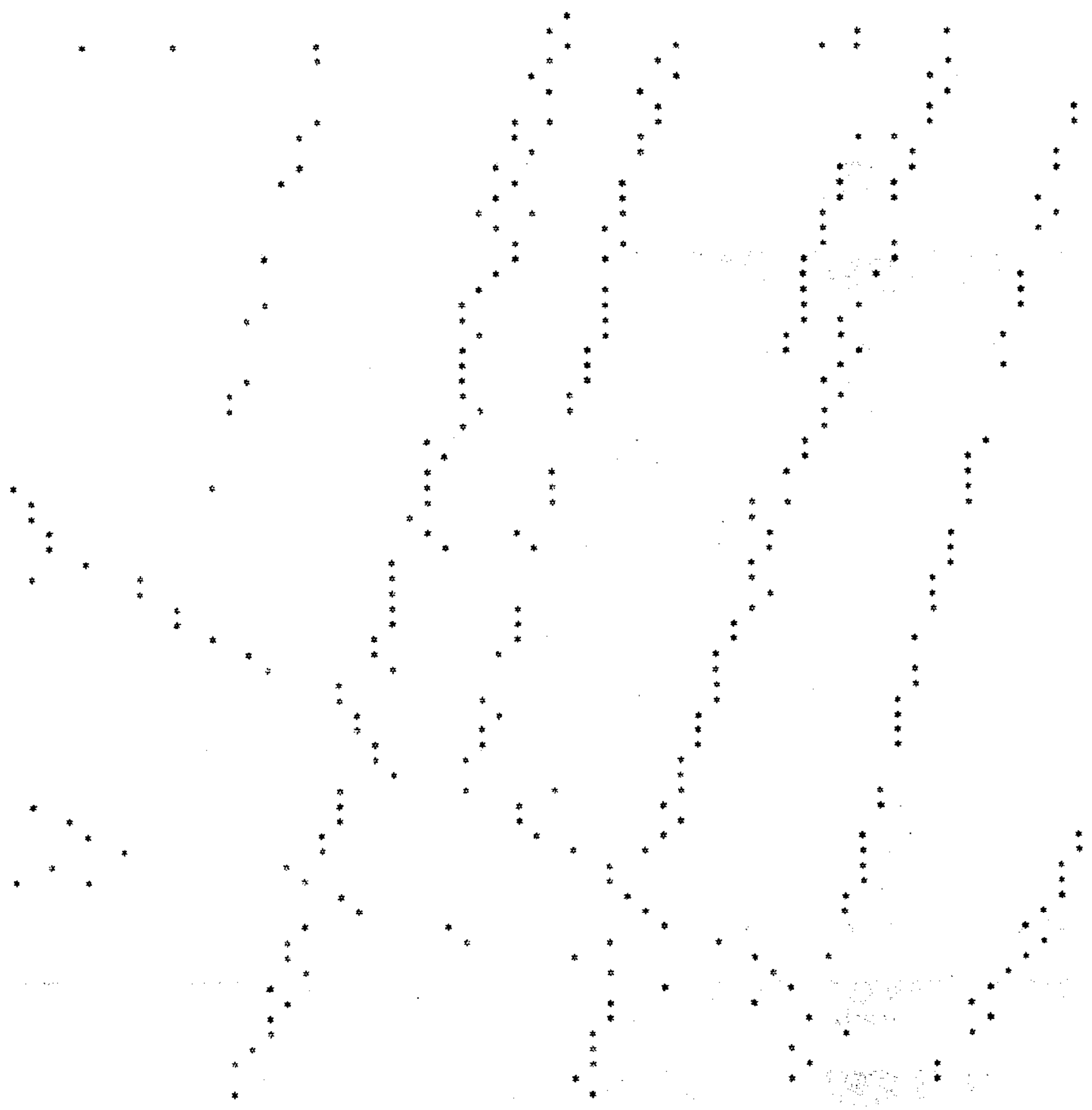


Figure 6.2(a) Bubble Chamber Track Digitizings

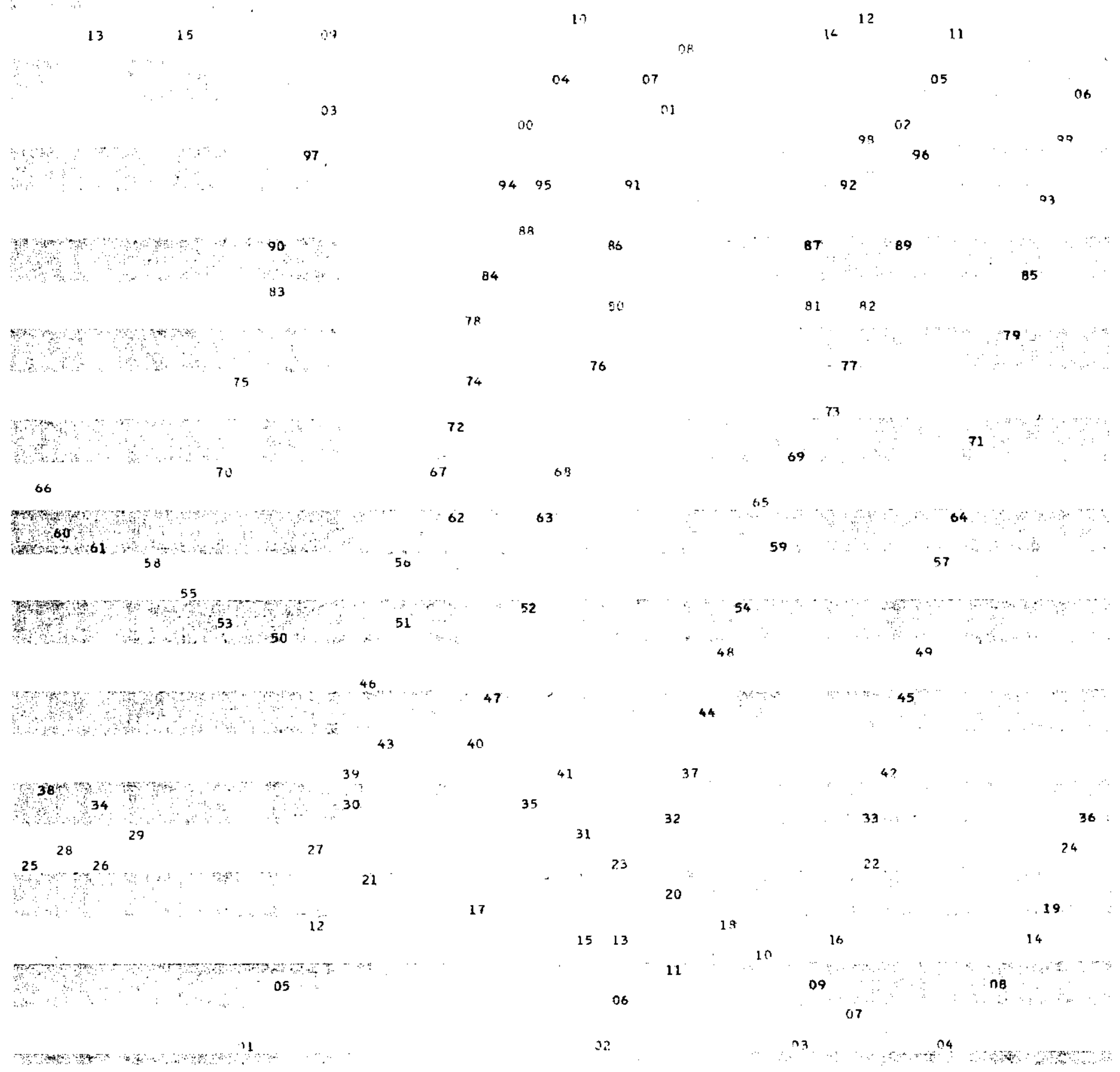


Figure 6.2(b) Results of Blobbing

```
TRIPLET AND CHAIN LIST..P1,P2,P3,THETA,CHAIN
    1          2          6         13        0.1405647E 01          21
    2          2          6         20        0.1212025E 01          -1
    3          2         11         18        0.8645967E 00          38
    4          4          7          9        0.2601173E 01          16
    5          4          8         14        0.9994586E 00          23
    6          4          8         19        0.1032961E 01          26
    7          4          9         10        0.2573761E 01          29
    8          4          9         18        0.2525295E 01          34
    9          4         10         18        0.2525295E 01          41
   10          4         14         19        0.1032961E 01          44
   11          1          5         12        0.1186633E 01          22
   12          1          5         21        0.1053314E 01          -1
   13          3         11         13        0.2546183E 01          -1
   14          3         16         22        0.1302093E 01          52
   15          3         20         23        0.2265534E 01          62
   16          7          9         10        0.2475645E 01          29
   17          7          9         18        0.2432966E 01          34
   18          7          9         20        0.2521343E 01          35
   19          7         10         18        0.2432966E 01          41
   20          7         10         20        0.2521343E 01          -1
   21          6         13         23        0.1496856E 01          -1
   22          5         12         30        0.1306308E 01          56
   23          8         14         19        0.1048393E 01          44
   24          8         14         24        0.1085174E 01          46
   25          8         14         36        0.1124690E 01          -1
   26          8         19         24        0.1085174E 01          61
   27          8         19         36        0.1124690E 01          -1
   28          8         24         36        0.1124690E 01          -1
   29          9         10         18        0.2414950E 01          41
   30          9         10         20        0.2539306E 01          -1
   31          9         10         23        0.2474900E 01          43
   32          9         16         22        0.1190289E 01          52
   33          9         16         33        0.1244638E 01          54
   34          9         18         23        0.2474900E 01          57
   35          9         20         23        0.2474900E 01          62
   36          9         22         33        0.1244638E 01          70
   37         11         15         17        0.2846151E 01          -1
   38         11         18         33        0.7392734E 00          -1
   39         11         20         32        0.1623381E 01          -1
   40         10         15         13        0.2982918E 01          -1
   41         10         18         23        0.2471941E 01          57
   42         10         18         31        0.2462920E 01          59
   43         10         23         31        0.2462920E 01          73
   44         14         19         24        0.1141033E 01          61
   45         14         19         36        0.1180188E 01          -1
   46         14         24         36        0.1180188E 01          -1
   47         15         13         18        0.1243550E 00          -1
   48         15         23         37        0.1069782E 01          -1
   49         15         32         37        0.1069782E 01          79
   50         13         31         41        0.1837049E 01          -1
   51         13         32         37        0.1227772E 01          79
   52         16         22         33        0.1297785E 01          70
   53         16         22         42        0.1304543E 01          72
   54         16         33         42        0.1304543E 01          82
   55         12         29         34        0.2553590E 01          77
   56         12         30         39        0.1337052E 01          85
   57         18         23         31        0.2498092E 01          73
   58         18         23         35        0.2498092E 01          -1
   59         18         31         35        0.2498092E 01          78
   60         17         35         41        0.1046000E 01          87
   61         19         24         36        0.1199903E 01          -1
   62         20         23         31        0.2356194E 01          73
```

Figure 6.2(c) A Partial List of the Collinear Triples

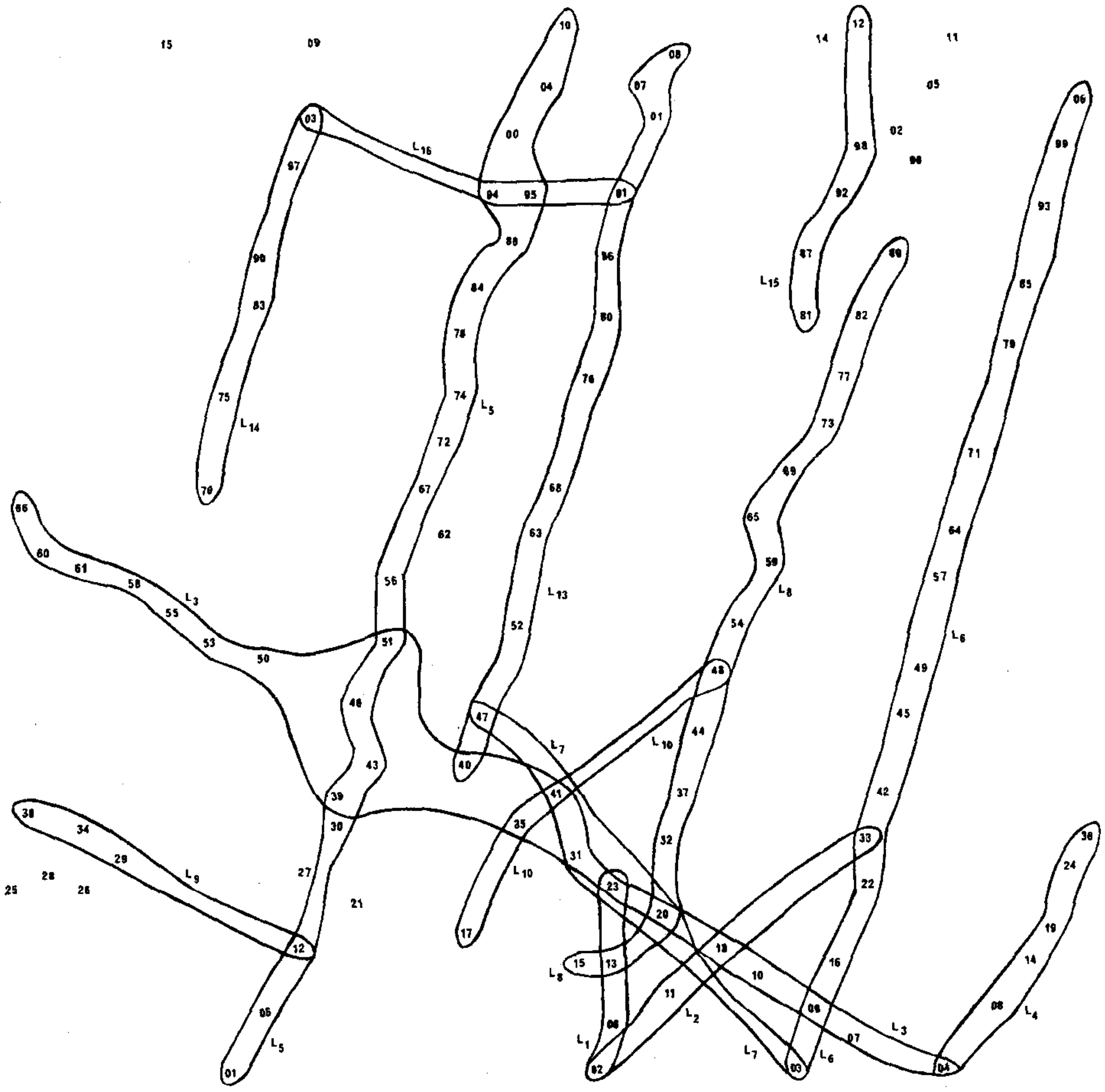


Figure 6.2(d) Lines Found By GPLR

193 co-linear triples were formed with $\Delta x = \Delta y = 12$ and $\epsilon = 0.05$. Figure 6.2(c) lists the first 61 of these. Column 1 is the triple number; columns 2, 3, and 4 contain the cell numbers comprising a triple; column 5 is the angle of the triple in radians. Column 6 represents the results of the chaining procedure. -1 indicates the end of a chain; otherwise, the next triple number in the chain appears. Thus, triple 5 starts the triple chain: (5, 23, 44, 61) .

The merging routine uses $\Delta\theta = 0.2$. The 14 lines that were found are labeled in Figure 6.2(d). (Lines consisting of only 3 points are not included.) Figure 6.2(e) lists these lines; for each line the following data appears from left to right across the page: line number, first point on line, coordinates of first endpoint, last point, coordinates of last point, number of cells on the line, and approximate angle in radians.

```
 LISTING OF THE   16 LINES FOUND BY MKLINE

LINE NO  FSTPT    X        Y       LSTPT    X        Y     NO OF PTS ANGLE
      1     2   32.75   953.50     23     34.33   966.00        4 0.1451251E 01
      2     2   32.75   953.50     33     48.33   969.33        4 0.8019347E 00
      3     4   52.00   953.50     66      1.67   991.00       22 0.2503973E 01
      4     4   52.00   953.50     36     60.00   969.00        6 0.1123087E 01
      5     1   13.33   953.67    110     31.50  1021.50       20 0.1344435E 01
      6     3   44.33   954.00    106     60.00  1017.00       16 0.1316285E 01
      7     3   44.33   954.00     47     27.33   977.00        6 0.2226089E 01
      8    15   32.00   961.00     89     49.67  1007.00       15 0.1240977E 01
      9    12   16.50   961.50     38      2.00   971.00        4 0.2577381E 01
     10    17   25.50   962.50     48     40.00   979.50        4 0.8868051E 00
     13    40   26.33   973.67    108     37.67  1020.00       12 0.1353413E 01
     14    70   12.00   992.00    103     18.00  1016.00        6 0.1336846E 01
     15    81   44.50  1002.50    112     48.00  1021.50        5 0.1333159E 01
     16    91   34.75  1010.50    103     18.00  1016.00        4 0.2877980E 01
```

Figure 6.2(e) Listing of the Lines Found by GPLR

```
CELL NOS AND THEIR LINES..NO,PLINE
     2  00000003     4  0000000C
     1  00000010     3  00000060
     7  00000004     6  00000001
     5  00000010     8  00000008
     9  00000024    11  00000002
    10  00000004    14  00000008
    15  00000080    13  00000081
    16  00000020    12  00000110
    18  00000006    17  00000200
    19  00000006    20  000000C4
    21  00000000    22  00000020
    25  00000000    26  00000000
    23  00000045    27  00000000
    24  00000008    28  00000000
    29  00000100    31  00000044
    32  00000080    36  00000008
    33  00000022    34  00000100
    30  00000010    35  00000204
    38  00000100    37  00000080
    39  00000014    41  00000244
    42  00000020    40  00001004
    43  00000014    44  00000080
    45  00000020    47  00001040
    46  00000014    48  00000280
    49  00000020    50  00000004
    51  00000014    53  00000004
    52  00001000    54  00000080
    55  00000004    58  00000004
    56  00000010    57  00000020
    59  00000080    61  00000004
    60  00000004    62  00000000
    63  00001000    64  00000020
    65  00000080    66  00000004
    67  00000010    70  00002000
    68  00001000    69  00000080
    71  00000020    72  00000010
    73  00000080    74  00000010
    75  00002000    76  00001000
    77  00000080    79  00000020
    78  00000010    80  00001000
    81  00004000    82  00000080
    83  00002000    84  00000010
    85  00000020    87  00004000
    86  00001000    90  00002000
    89  00000080    88  00000010
    93  00000020    91  00009000
    92  00004000    95  00008010
    94  00008010    96  00000000
    97  00002000    98  00004000
    99  00000020   100  00000010
   102  00000000   101  00001000
   103  0000AC00   106  00000020
   105  00000000   104  00000010
   107  00001000   108  00001000
   109  00000C00   113  00000000
   115  00000000   114  00000000
   111  00000000   110  00000010
   112  00004000
```

Figure 6.2(f) Cells and Their Lines

Note that there are no lines numbered 11 or 12 due to the final merging phase of GPLR. Figure 6.2(f) lists the lines, if any, that each cell is part of; each bit in the hexadecimal word following the cell number represents a line; if there is a "1" in bit i, i = 0, 31, that cell is on line i+1, where the bits of a word are numbered from 0 to 31 starting from the low order bit. Thus cell 41 is on lines 3, 7, and 10 .

Several comments can be made on the results of this analysis (Figure 6.2(d)):

1. All obvious lines were found.
2. Some extraneous lines, such as L_2, L_9, L_{10}, and L_{16} were included. By restricting lines to consist of at least 5 cells, these are eliminated (along with L_1).
3. L_3, L_5, L_6, and L_8 traverse successfully across confused areas and intersect other lines.
4. The extension of L_8, through cells 96, 105, and 111 at its top end, and through cells 6 and 2 at its bottom, was missed.
5. L_7 could be merged with L_3 by increasing $\Delta\theta$ to 0.3 .

This example can be considered a "worst case" for GPLR.

6.4 BLANK AND DON'T CARE PRIMITIVES

Since the concept of blank and don't care primitives originated as part of the PDL system, their recognition is discussed only in the context of the PDL picture parser.

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

$$P \rightarrow (r + b + S)$$

$$S \rightarrow s1 | s2 | \ \ldots \ | sn, \qquad n > 1$$

where r, s1, s2, ..., 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 vacuous and always return <u>true</u>.

<u>Example 1</u>:

Consider the primitive ep in the particle physics syntax of Figure 4.2. Its purpose is to find the tail of some member of $\mathcal{P}(TM)$ entering at the left edge of the picture. This can be done easily by using GPLR locally in regions near the left edge to find the beginning of a track cm; once this is accomplished, the recognizer for cm can determine its extent, curvature, and perhaps, point density.

<u>Example 2</u>:

The page recognition syntax of Figure 4.5 employs the primitive ics as the inter-character spacing for letters in a word; this spacing is variable within certain limits, depending on the printing process and the letters. The recognition routine for ics has only to find the beginning (tail) of a letter; this can be done by counting point densities along a line or by some other simple global test. The primitive recognizers for the letters do the actual classification, but their location has been previously discovered by ics .

Blank primitives are used extensively in the application of the next chapter. A variety of methods are employed, depending on the severity of the geometrical constraints defined by the blank primitives and the classes of primitives concatenated onto them.

CHAPTER 7

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 by Barber, Gittleman, O'Neill, and Richter [1965, 1966]. This film provided an excellent test--in a real and non-trivial setting--of the approach to picture processing that has been advocated and developed in the preceding chapters.

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 is observed via a set of spark chambers and counters through which the electrons then pass. Figure 7.1 shows the chamber-counter geometry. Each scattered electron traverses successively through a 6 gap chamber, a 4 gap chamber, and a shower chamber; the possible points of interaction (collision) lie along the horizontal "median" line in the center of the figure.

30,000 photographs were taken of two views of the chamber and a data box; a mirror arrangement was used to capture each information set on one frame. 400 of the photographs contained "events" of interest. The film has been manually analyzed and the results subjected to a further computer analysis to obtain the angular distributions.

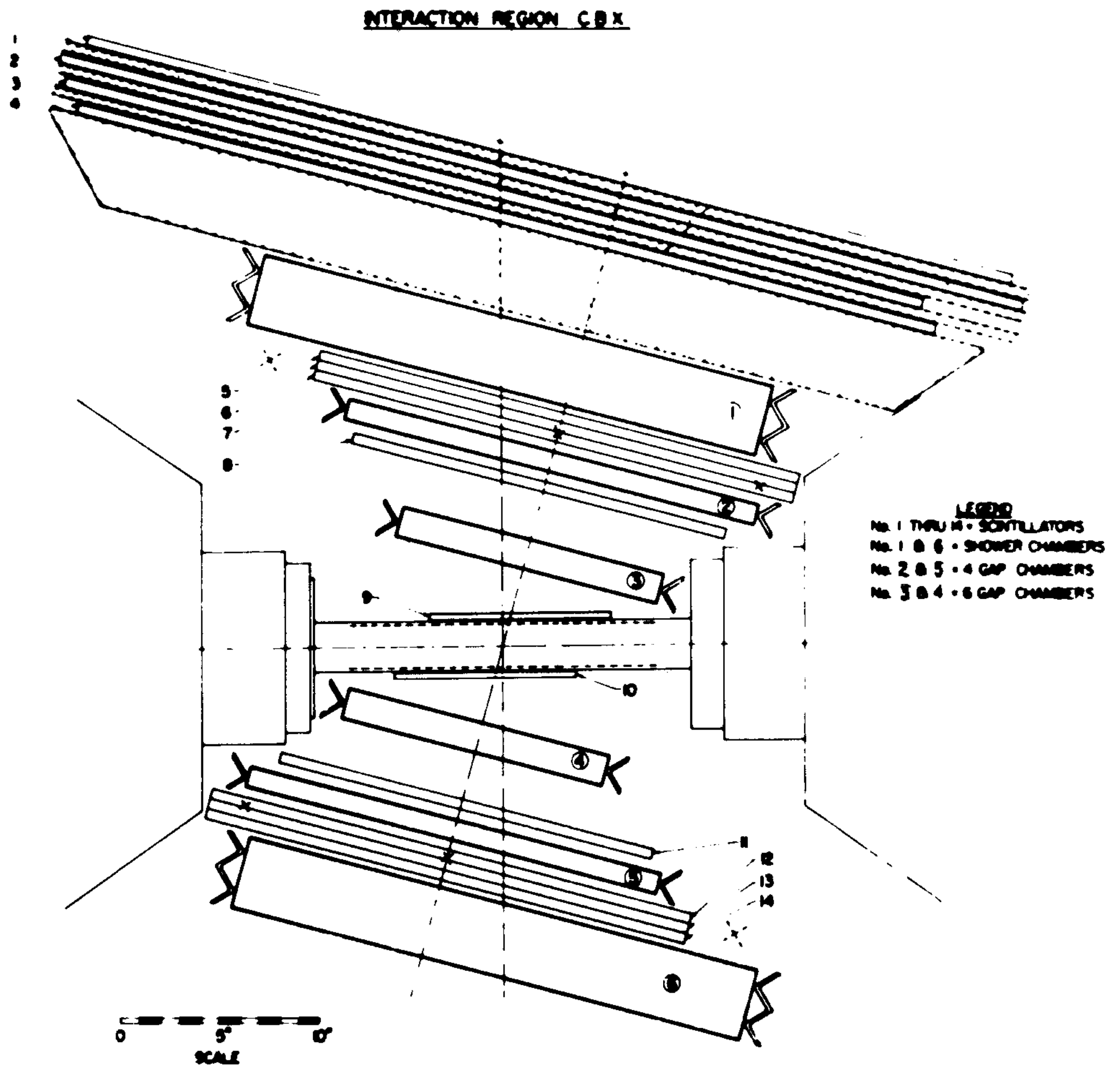

INTERACTION REGION C B X
1
2
3
4
5
6
7
8
9
10
11
12
13
14
1
2
3
4
5
6
LEGEND
No. 1 THRU 14 • SCINTILLATORS
No. 1 & 6 • SHOWER CHAMBERS
No. 2 & 5 • 4 GAP CHAMBERS
No. 3 & 4 • 6 GAP CHAMBERS
0
5"
10"
SCALE

Figure 7.1 Chamber-Counter Geometry

7.1 CHARACTERISTICS OF THE PICTURES

A schematic of the film format is illustrated in Figure 7.2. 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 a 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.

Figure 7.3 is an actual photograph of a cosmic ray shower passing through the chambers. It illustrates the appearance of sparks and the relative locations of the components on the film. In general, there is considerable variation across the film; some of the fiducials, sparks, and characters are clear, whereas others can barely be seen.

An event appears in the photograph of Figure 7.4(a) as indicated by the collinear sparks in the 4 and 6 gap chambers. This picture was digitized by the Hummingbird and displayed on the 2250; Figure 7.4(b) is a photograph of the scope during the display. Because of the Hummingbird characteristics discussed earlier, the digits are not recognizable in most cases; the digitizings obtained for the remaining

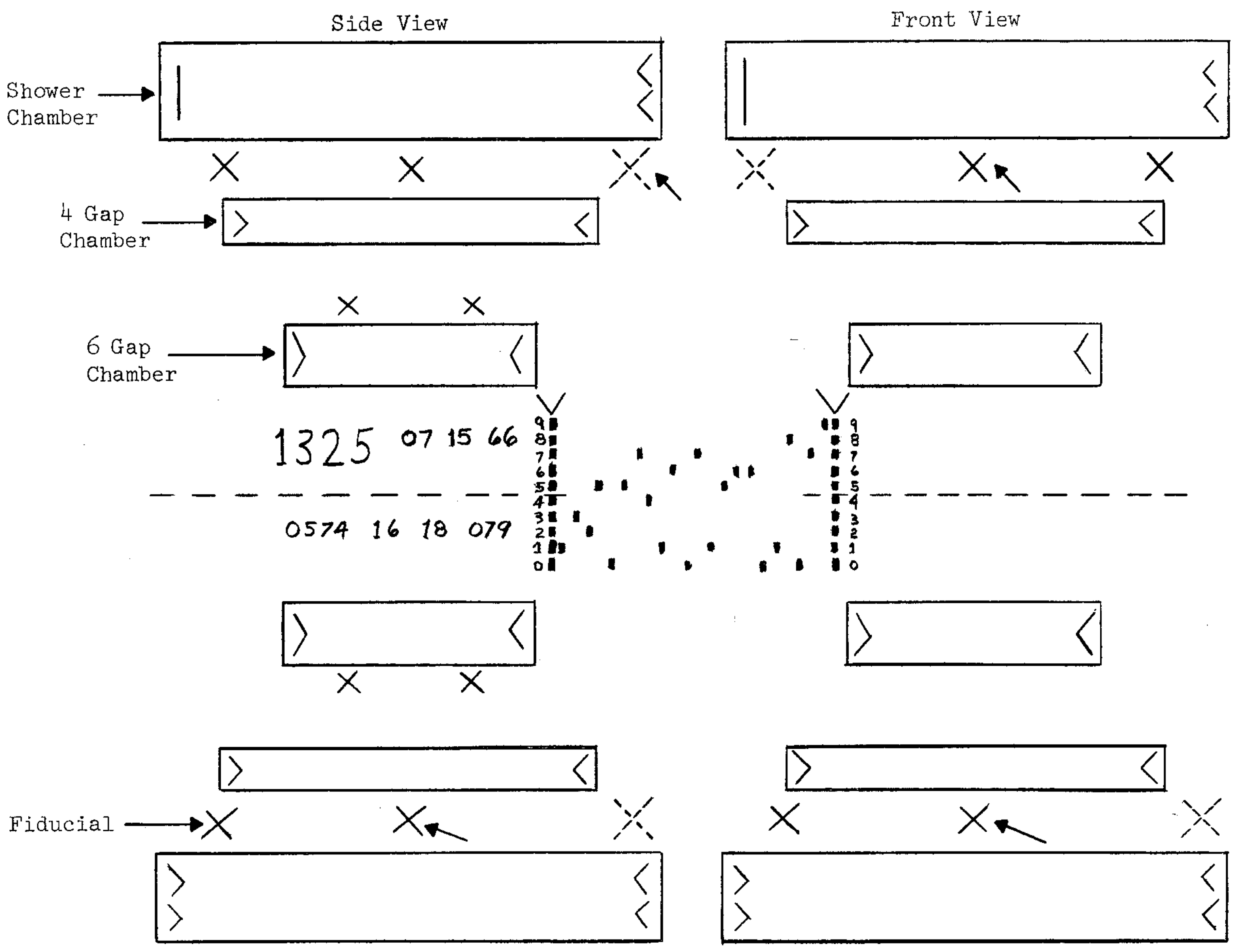


Figure 7.2 Film Format

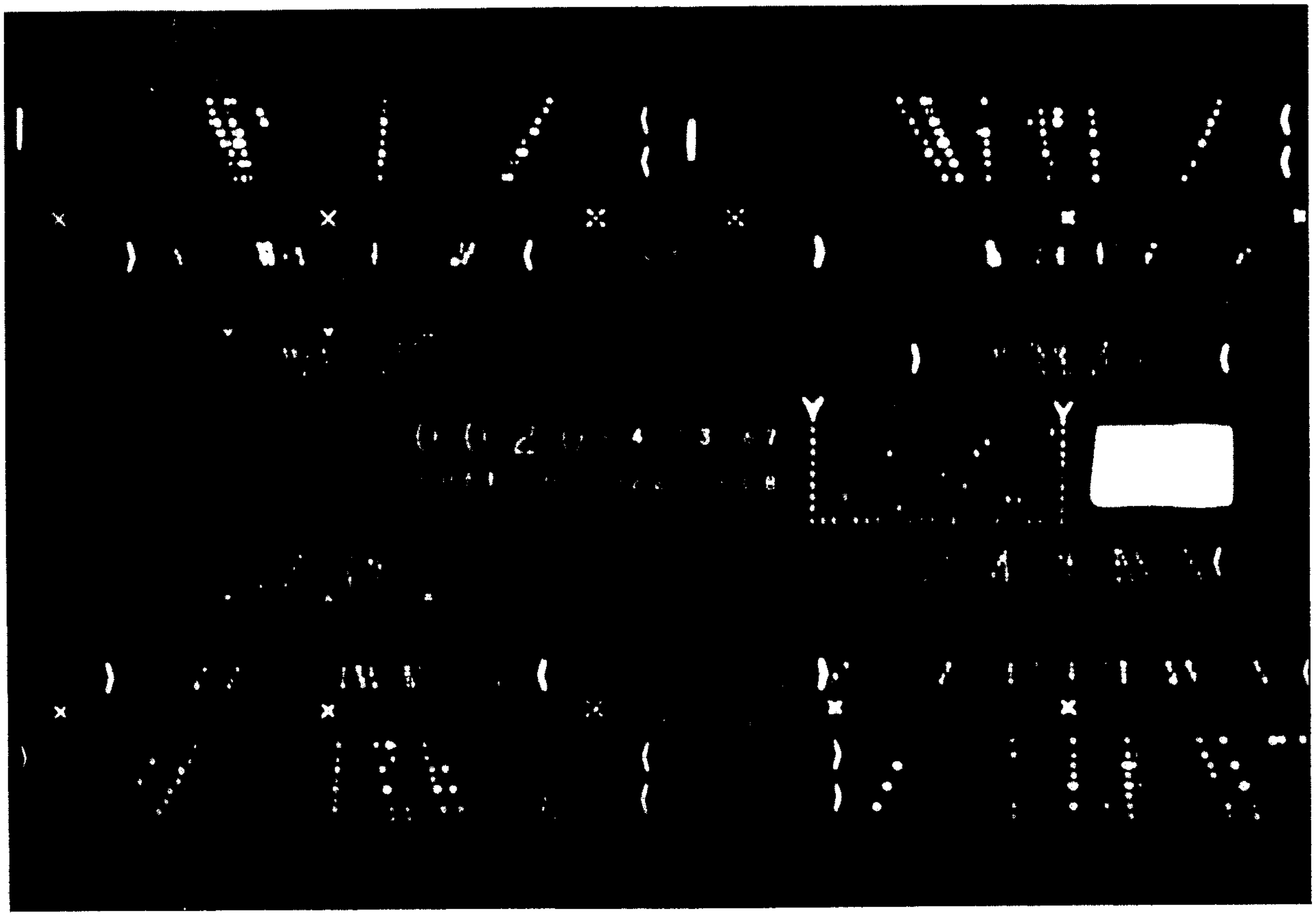


Figure 7.3 Cosmic Ray Shower

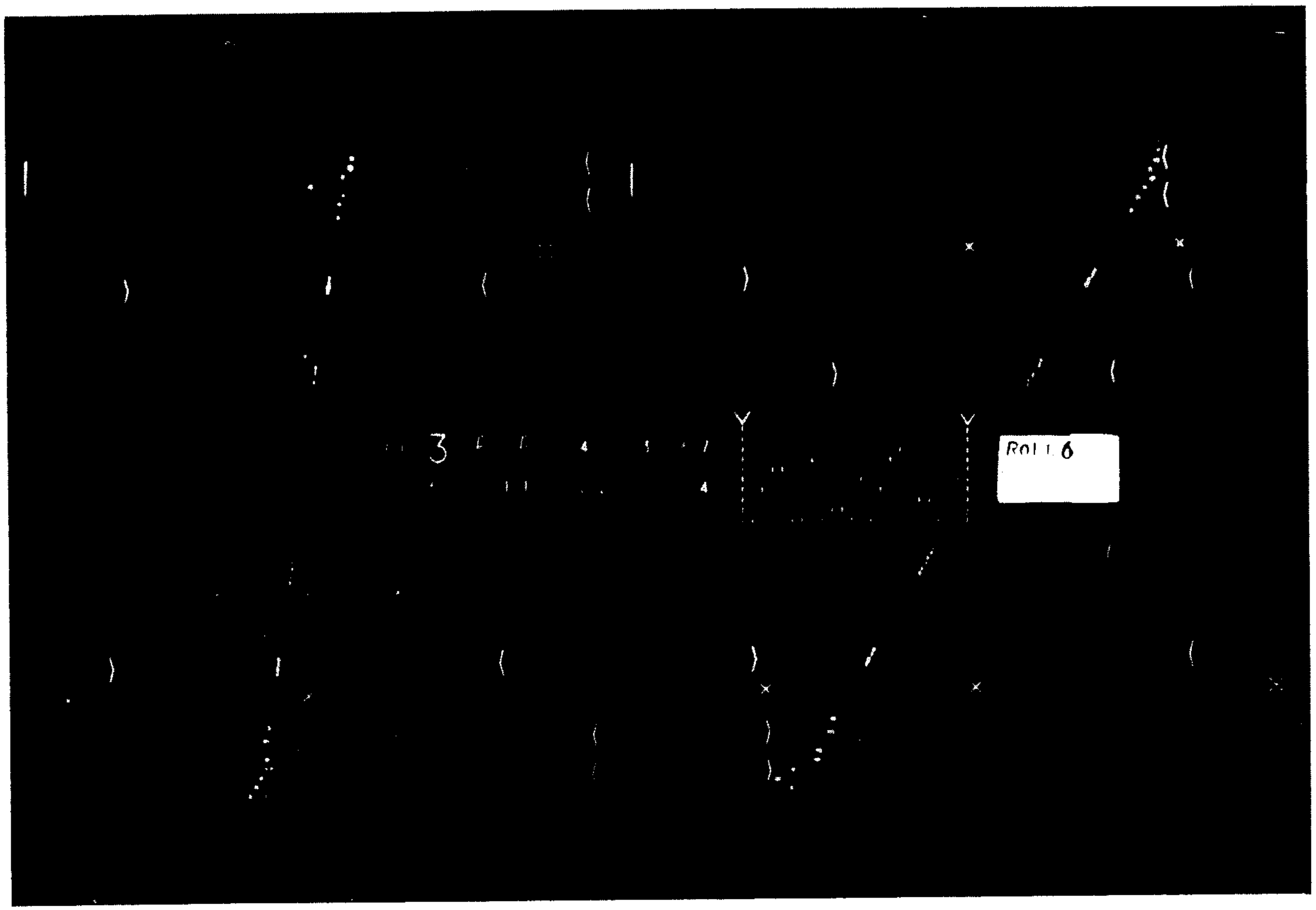


Figure 7.4(a) Photograph Containing a Typical Event - Frame 355

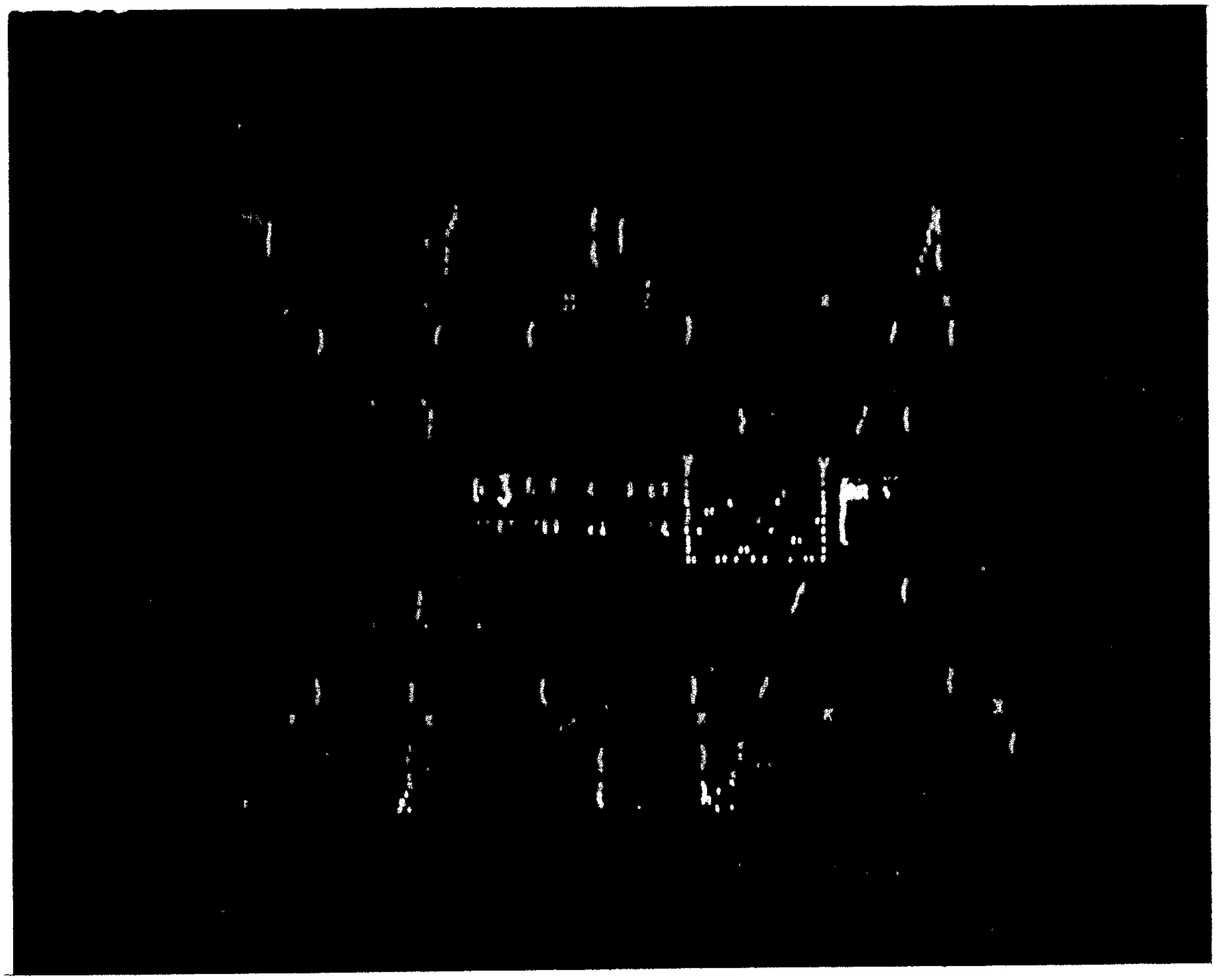


Figure 7.4(b) Digitized Version of (a)

portions of the picture accurately portray the original. Printer plots of the digitizings in several small windows of another member of this picture class are shown in Figure 7.5(a)-(d); these are typical areas viewed by the primitive recognizers. The isolated digitizings in Figure 7.5(a) are background 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 non-uniformity of sparks, slipping of the camera mirrors between frames, occasional malfunction of parts of the data box, errors of digitization, 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.

7.2 THE GRAMMAR

Figure 7.6 lists the primitive names, the non-terminal symbols, and the grammar defining the picture class for the colliding beam experiment

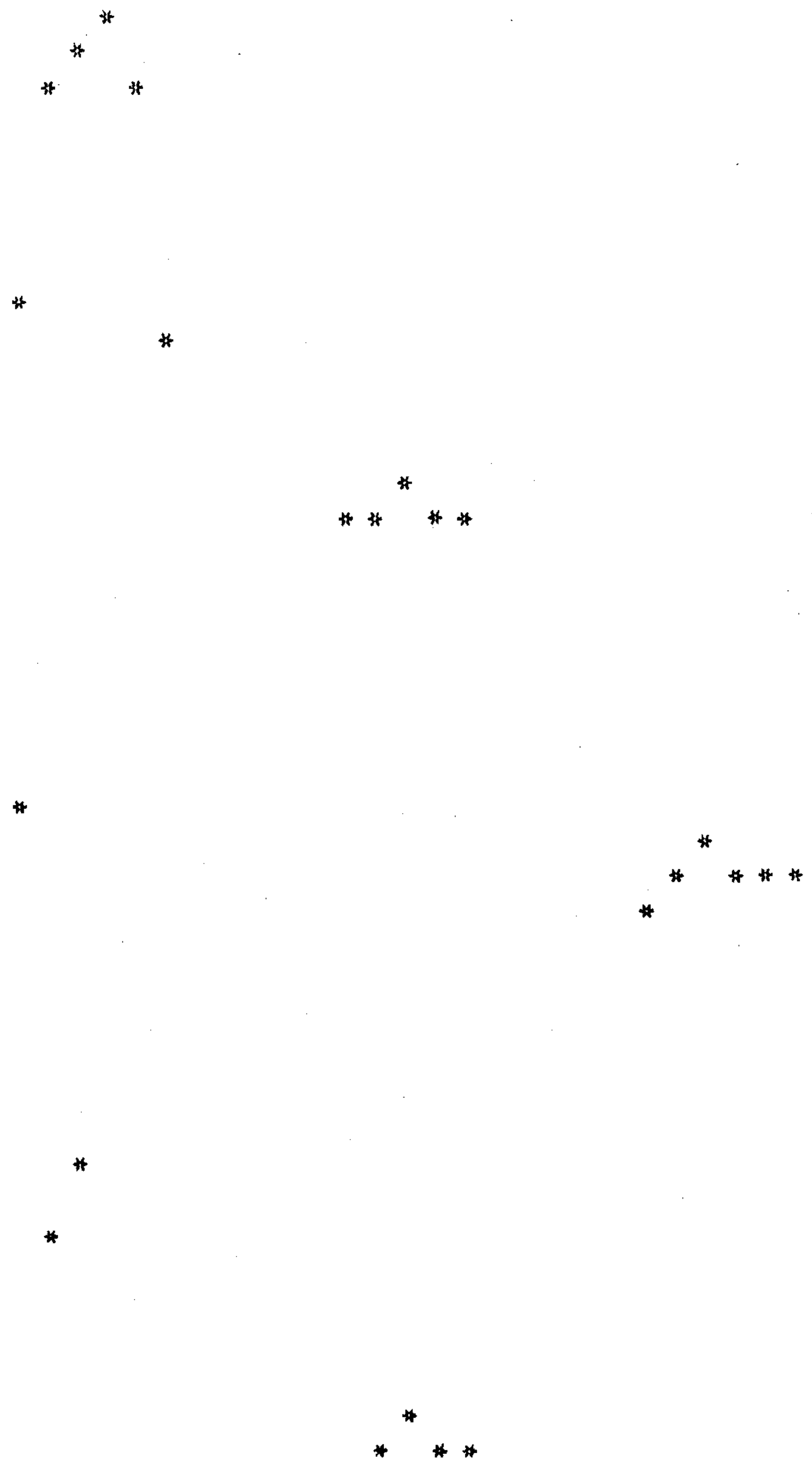


Figure 7.5(a) Plot of Digitized Sparks

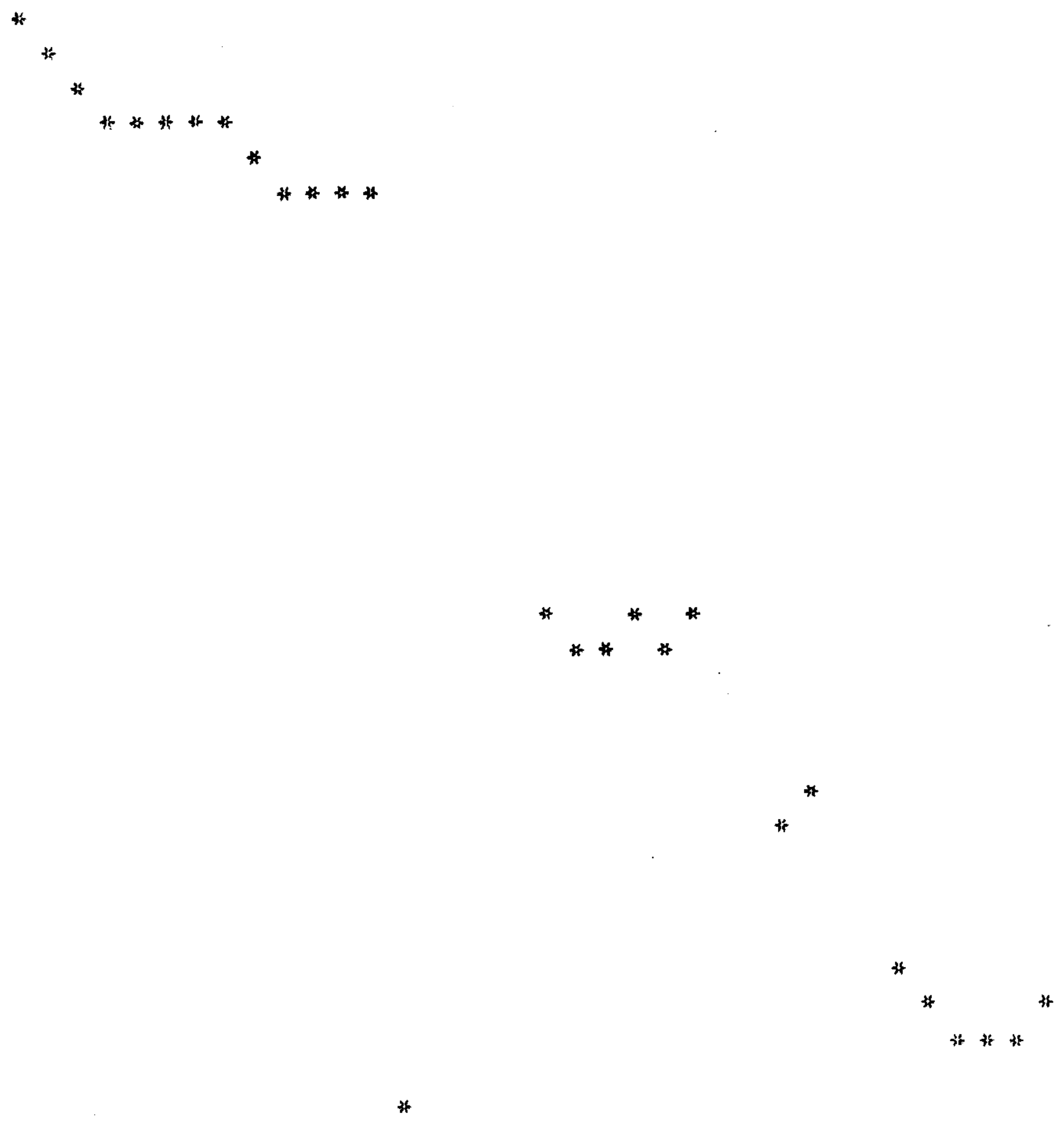


Figure 7.5(b) A Collinear Set of Sparks

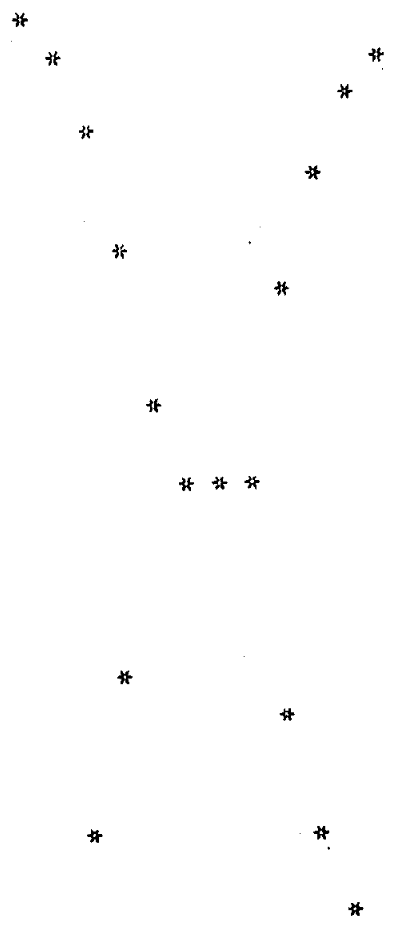


Figure 7.5(c) Plot of a Fiducial

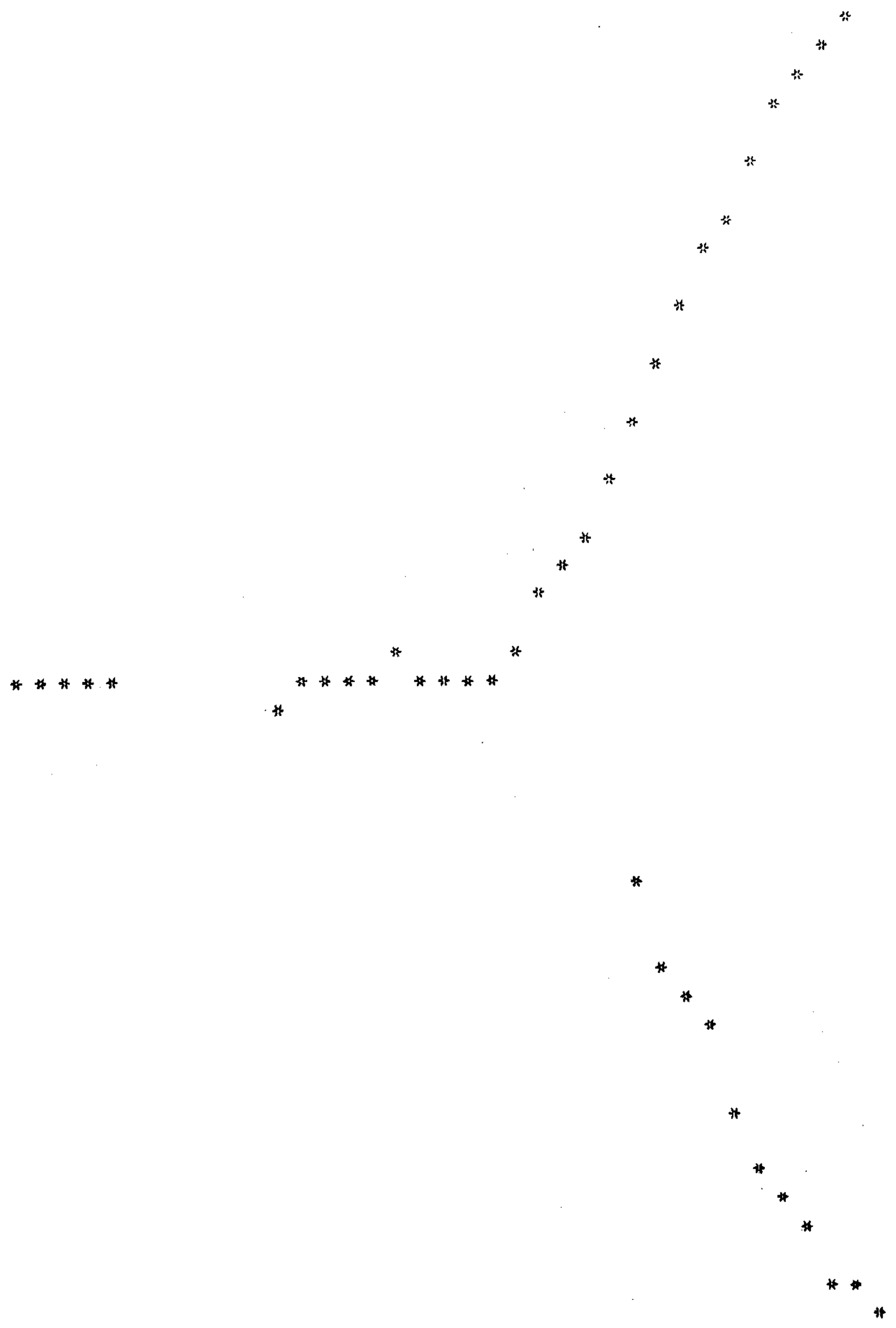


Figure 7.5(d) Top Part of Data Box Boundary

Figure 7.5 Printer Plots of Selected Digitized Areas

```
NPRIM= 23 PRIM  PRSUB

          BBND       2            1
          B4GH      21            2
          B4GL      12            3
          B6GH      22            4
          B6GL      13            5
          DIGT       4            6
          EVNH      23            7
          EVNT      14            8
          IDS        3            9
          NOFB      11           10
          NOFT      20           11
          NOSB      10           12
          NOST      19           13
          NULL       1           14
          SHFB      18           15
          SHFT      16           16
          SHSB      17           17
          SHST      15           18
          STRT       5           19
          XFB        8           20
          XFT        9           21
          XSB        6           22
          XST        7           23
```

Figure 7.6(a) Primitive Class Names

```
NPROD= 24NON-TERMINALS--
                           COB            1
                           CFB            2
                           CFT            3
                           CHI            4
                           CLBM           5
                           CLO            6
                           CSB            7
                           CST            8
                           DATA           9
                           FB46          10
                           FRVW          11
                           FT46          12
                           HI4           13
                           HI4D          14
                           HI6           15
                           HI6D          16
                           LO4           17
                           LO4D          18
                           LO6           19
                           LO6D          20
                           SB46          21
                           SDVW          22
                           SPRK          23
                           ST46          24
```

Figure 7.6(b) Non-Terminal Symbols

```
CLBM= (STRT + ( CDB . (FRVW . SDVW ) ) )
CDB =  (BBND . (IDS + (DATA + (IDS + BBND) )) )
DATA=   ( DIGT.(IDS + DATA ))
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
SB46=   (L04.L06)
FB46=   NOFB
FB46=   (L04.L06)
L04 =   ((B4GL+CLO).L04D)
L04 =   NULL
L04D=   L04
L04D=   NULL
CLO =   EVNT
CLO =   SPRK
SPRK=   NULL
L06 =   ((B6GL+SPRK).L06D)
L06 =   NULL
L06D=   L06
L06D=   NULL
ST46=  NOST
ST46=  (HI6 .HI4)
FT46=  NOFT
FT46=  (HI6 . HI4)
HI4 =  ((B4GH+SPRK).HI4D)
HI4 =  NULL
HI4D=  HI4
HI4D=  NULL
CHI =  EVNH
CHI =  SPRK
HI6 =  ((B6GH + CHI ).HI6D )
HI6 =  NULL
HI6D=  HI6
HI6D=  NULL
$   =
```

Figure 7.6(c) Grammar for Colliding Beam Pictures film; next to each primitive name is the number of its recognition subroutine. There are 23 primitive class names, 24 non-terminal symbols, and 39 productions. 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

$$\left\{\begin{matrix}\text{SD}\\ \text{FR}\end{matrix}\right\}\text{VW}:\ \left\{\begin{matrix}\underline{\text{s}}\text{ide}\\ \underline{\text{f}}\text{ront}\end{matrix}\right\}\ \text{vie}\underline{\text{w}}$$

$$\text{C}\left\{\begin{matrix}\text{ST}\\ \text{FT}\\ \text{SB}\\ \text{FB}\end{matrix}\right\}:\ \underline{\text{c}}\text{ontents of}\left\{\begin{matrix}\underline{\text{s}}\text{ide}\\ \underline{\text{f}}\text{ront}\\ \underline{\text{s}}\text{ide}\\ \underline{\text{f}}\text{ront}\end{matrix}\right\}\text{view chamber,}\left\{\begin{matrix}\underline{\text{t}}\text{op}\\ \underline{\text{t}}\text{op}\\ \underline{\text{b}}\text{ottom}\\ \underline{\text{b}}\text{ottom}\end{matrix}\right\}\text{half of picture}$$

$$\left\{\begin{matrix}\text{ST}\\ \text{FT}\\ \text{SB}\\ \text{FB}\end{matrix}\right\}46:\left\{\begin{matrix}\underline{\text{s}}\text{ide}\\ \underline{\text{f}}\text{ront}\\ \underline{\text{s}}\text{ide}\\ \underline{\text{f}}\text{ront}\end{matrix}\right\}\text{view,}\left\{\begin{matrix}\underline{\text{t}}\text{op}\\ \underline{\text{t}}\text{op}\\ \underline{\text{b}}\text{ottom}\\ \underline{\text{b}}\text{ottom}\end{matrix}\right\}\text{half, }\underline{4}\text{ and }\underline{6}\text{ gap chambers}$$

$$\left\{\begin{matrix}\text{LO4, LO4D}\\ \text{LO6, LO6D}\\ \text{HI4, HI4D}\\ \text{HI6, HI6D}\end{matrix}\right\}:\ \left\{\begin{matrix}\underline{\text{l}}\text{ow}\\ \underline{\text{l}}\text{ow}\\ \underline{\text{h}}\text{igh}\\ \underline{\text{h}}\text{igh}\end{matrix}\right\}\text{half of picture,}\left\{\begin{matrix}\underline{4}\\ \underline{6}\\ \underline{4}\\ \underline{6}\end{matrix}\right\}\text{gap chamber}$$

$$\text{C}\left\{\begin{matrix}\text{LO}\\ \text{HI}\end{matrix}\right\}:\ \text{either a spark in a}\left\{\begin{matrix}\underline{\text{l}}\text{ow}\\ \underline{\text{h}}\text{igh}\end{matrix}\right\}\text{4 gap chamber or an event}$$

SPRK: a spark

The parsing sequence defined by the grammar proceeds as follows: The data box is first found (since it is obvious and an 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 fiducial X's; the chambers are located in fixed positions relative to these. An arrow in Figure 7.2 points to each of the 4 fiducials used. In each of the four parts, or quadrants, or 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, EVNH) . 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.

7.3 PRIMITIVE RECOGNIZERS

GPLR and the blobbing routine form the basis of all recognition functions. The picture components are grouped into meaningful structures by blank primitives; the interaction between these and the primitive classes to which they may be concatenated is noted in the descriptions below. The picture origin is at the lower left-hand corner, and has the coordinates (1, 1) .

1. Starting the Analysis

The recognition routine for STRT looks in a large centrally-located window for a long vertical line, representing the left boundary of the data box. The head of STRT is taken at the lowest (smallest y-coordinate) point on this line.

2. The Data Box

Most of the recognition work for the data box boundaries (BBND) is done by its preceding blank primitive (STRT or IDS); the BBND routine verifies the length and angle of the line. The digits (DIGT) in the data box are separated horizontally by an interdigit space (IDS) of 29 ± 4 raster units. IDS retrieves a small rectangular window around the expected location of the next digit and constructs blobs; each blob is the size of a digit. The head of IDS is (x, y) where x is the x-coordinate of the lowest cell found, if any, and y is the y-coordinate of the tail of IDS . It is possible for 0 to 5 digits to appear at any position; occasional malfunction of the data box results in either 0 or more than 1 digit. If IDS finds no cells, a nominal value is given to x and a flag is added to its value list. The DIGT routine returns true if less than 6 cells are found in the vertical strip above the head of IDS; otherwise false is returned and the right boundary of the data box has probably been reached. DIGT computes the actual digit represented by the location of each blob and puts this in the value list. The head of DIGT is set to the coordinates of its highest (largest y-coordinate) digit. The relationship between IDS and DIGT is a good example of the role of a blank primitive: IDS determines if something is in the window; DIGT does the detailed recognition.

3. Fiducial Finding Routines

The approximate location of the fiducials relative to the tail of the data box is known (within about 60 raster units in each of the x and y directions); their accurate positions must be determined in order to allow precise reconstruction of the picture in the 3-dimensional space of the physics experiment. The tail of each fiducial primitive is the tail of the data box CDB; their heads are defined as the intersection point of the two arms of the "X"; this is a case where it is convenient to define a primitive in terms of a don't care part (from the data box to the X) and a visible part (the X). SFB, SFT, and XSB are computed by a common routine which finds two intersecting line segments satisfying length and angle tolerances. XST, the dotted fiducial, appears as four symmetric blobs in the picture and a separate routine calculates the center of these.

4. Event and Spark Recognition in the 4 and 6 Gap Chambers

Each fiducial center is used as the tail for analysis of a quarter of the picture. The routines defining the primitives NOSB, NOFB, NOST, and NOFT retrieve the contents of the 4 and 6 gap chambers in each quarter. If the chambers are empty, true is returned. Otherwise, cells and lines are constructed by GPLR in each chamber and false is returned; the cell parameters are chosen so that each cell represents a spark. B4GL, B6GL, B4GH, and B6GH are blank primitives that point to the next spark; if the chamber is empty, false is returned. Each spark is then recognized as the first spark of an event (EVNT or EVNH) or an isolated spark (SPRK $\rightarrow$ NULL); EVNT and EVNH represent events in the lower and upper chambers respectively. An event is defined as a line (at least three collinear sparks) in the 6 gap chamber that is collinear with at least one spark in its corresponding 4 gap chamber. The number of sparks and the line angle is computed for the value list of an event. After an event or isolated spark is recognized, the sparks comprising the pattern are eliminated and the blank primitives above will be called again by the parser.

5. The Shower Chambers

The routines for the shower chamber primitives (SHST, SHSB, SHFT, SHFB) compute and return the coordinates of all sparks in these chambers.

The least satisfying, and most tedious and time-consuming aspects of the design of the primitive recognizers were the selection of the proper parameters for blob construction and GPLR, and the determination of the relative locations of the picture components. This was done by a detailed manual analysis of a printer plot of one picture. In any future developments, it is clear that an on-line interactive graphics system should be written to perform this task. This would provide a much more efficient and accurate means for setting and testing the parameters of the recognizers.

7.4 RESULTS OF THE PARSE

Pictures were digitized by the Hummingbird and stored on magnetic tape for later analysis by the SPDL system. One picture was used to debug the grammar and primitive recognizers, and set the recognition routine parameters. Nine additional frames were then analyzed successfully.

Eight of these frames were selected at random from those containing events; the remaining frame was arbitrarily chosen. Some adjustments were made for the event recognition parameters and the chamber location coordinates after the first run of the nine frames.

Figure 7.7 contains a photograph of the display after the analysis of the picture of Figure 7.4. The display contains an abstracted version of what has been recognized; each primitive, including blanks, is represented by a straight line segment terminating on the tail and head coordinates of the primitive. "T" and "H" indicate the tail and head of the last primitive found. Most of the primitive classes are identified by name in the figure. Appendix A contains 2250 display photographs of the remaining nine pictures, before the analysis and of their abstracted plot after the analysis. A listing of the natural semantics and parsing tree for frame 355 (Figures 7.4 and 7.7) is also included.

The contents of each data box was correctly identified in all frames except one, including several missing positions and a multiple digit position; in frame 406, the 4th digit (6) had too few digitizings and was indistinguishable from the background. The number of sparks recognized in each chamber was either correct, or one more than the correct number as counted from the original photographs; the deviation was due to the digitization process rather than the recognition mechanism. All obvious events were found. In some frame quarters, lines in the 4 and 6 gap chambers that constituted events were not collinear within tolerance, or there were too few sparks in the 6 gap chamber to form a line (less than 3), and the sparks were then considered isolated; this occurred three times (see frames 312, 356, and 414 in Appendix A). More

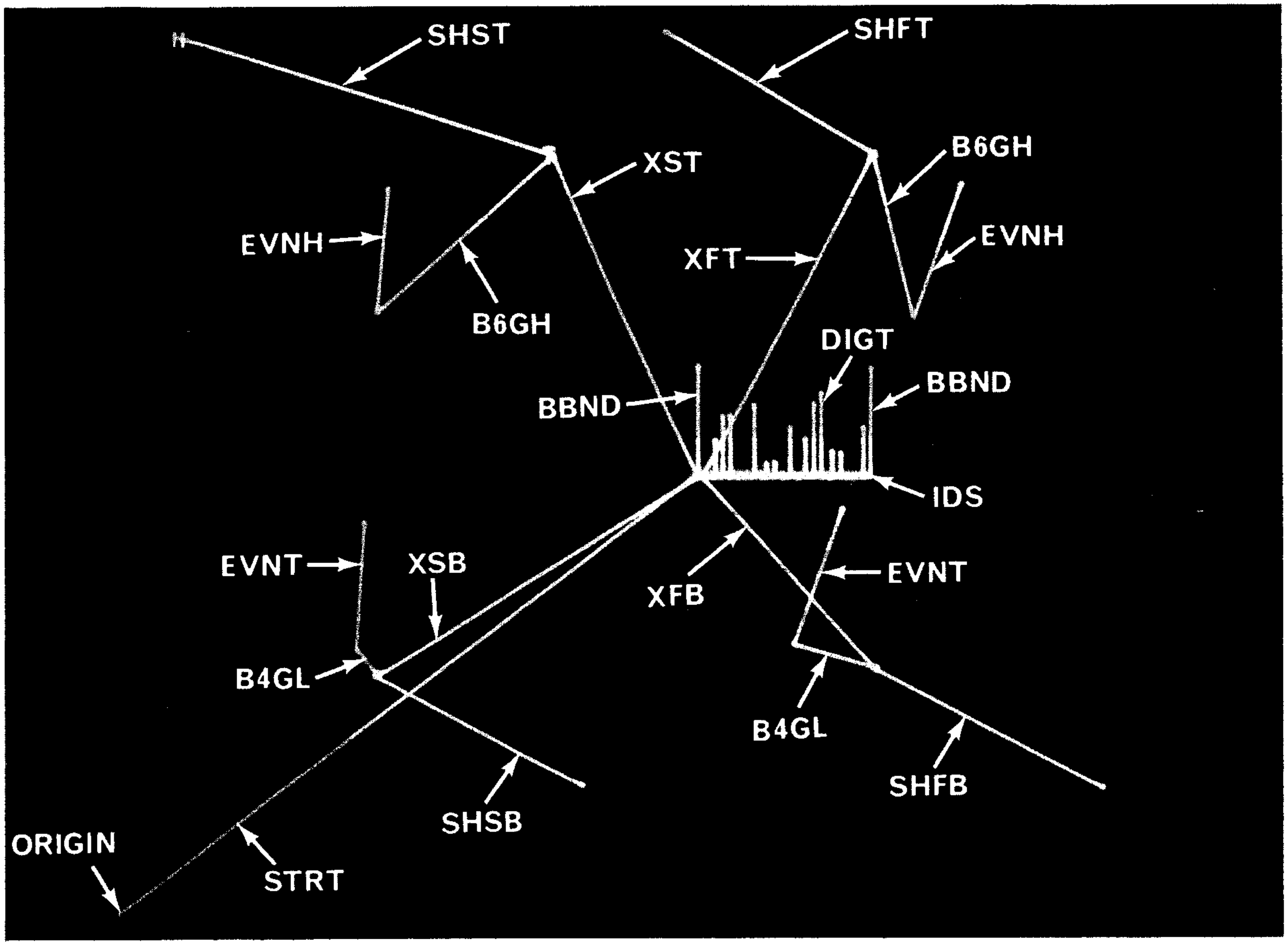


Figure 7.7 Analysis of Frame Number 355

complex logic in the event recognition routines could handle these cases; however, a study of the instructions for manual recognition that are given to technicians indicates that in ambiguous situations such as these, there is often no clear-cut decision procedure (Gittelman [1967]).

Approximately 1800 hits or digitizings were obtained for each picture. The number of non-blank primitives recognized in the nine frames is large:

22 x 9 = 198	data box digits (DIGT)
2 x 9 = 18	data box boundary elements (BBND)
4 x 9 = 36	fiducials (XST, XSB, XFT, XFB)
26	events (EVNT, EVNH)
249	sparks comprising events
173	isolated sparks (SPRK and SHST, SHSB, SHFT, SHFB)

Table 7.1 contains the results of a timing run. Times were computed with the interval timer on the IBM 360 (model 50); this timer counts in 60ths of a second. The "Total" column gives the total time elapsed (in seconds) from the start of the analysis (exclusive of tape reading time) until its completion for each frame. "Creation of Data Structure" contains the time required to convert the raw digitizings into the data structure used by the primitive recognizers. The parsing or analysis time is broken into two distinct parts--goal administration, which is everything except primitive recognition, and primitive recognition.

The most interesting and satisfying data gleaned from Table 7.1 are the relatively small amount of time the system spends in goal administration (less than 5% of the total parsing time) and the primitive recognition times. The hard work and "guts" of picture analysis lies in

SPDL ANALYSIS OF COLLIDING BEAM EXPERIMENT FILM

TIMING DATA (In Seconds)

Frame Number	Creation of Data Structure	Parsing		Total
		Goal Administration	Primitive Recognition	
93	0.63	0.18	6.33	7.14
312	0.65	0.43	6.94	8.02
355	0.65	0.28	5.87	6.80
356	0.60	0.32	6.07	6.99
375	0.68	0.30	7.15	8.13
403	0.65	0.32	6.10	7.07
406	0.62	0.23	5.97	6.82
414	1.00	0.35	6.80	8.15
416	0.58	0.18	5.57	6.33
Totals	6.06	2.59	56.80	65.45
Average Times	0.67	0.29	6.31	7.27

Table 7.1

primitive recognition and it is appropriate that a very high percentage of processing time is devoted to this task. One can conclude that, for this application, a simple goal-oriented parse is efficient. While the programs were written with clarity in mind rather than efficiency, the analysis times compare favorably with those of "one-of-a-kind" hand-coded systems (Brown [1967], Miller [1967], Dickens [1967b]). In order to use the colliding beam programs in a production system, some additions must be made. The distortions introduced by the digitization process and the photography must be eliminated; the former is usually accomplished by a calibration program (Brown [1966]). The patterns formed by SPDL analysis must then be reconstructed in real space. Finally, the results from all the frames must be accumulated and subjected to statistical analysis.

Another significant feature of this application was the short period of time required for the implementation. It took less than $1\frac{1}{2}$ man-months for the development of the colliding beam grammar, the writing and debugging of the primitive recognizers, and the picture analysis. The reasons for this efficient implementation are:

1. Each primitive recognizer could be treated almost independently of the others due to the SPDL environment in which it is used.
2. 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.
3. The visual display of the parse, especially during backtracking, enabled fast on-line detection of bugs.
4. All primitive recognizers used the general purpose blobbing and GPLR routines.

The sample colliding beam pictures were successfully analyzed without compromising the picture processing model or the PDL system. This application demonstrates the usefulness and potential of the formal description and parsing methods.

CHAPTER 8

CONCLUSIONS

A useful paradigm satisfies two criteria (Kuhn [1962]):

1. It must allow a better solution, in some sense, to a set of problems that can be obtained outside of the model. (The paradigm is clearly most exciting when the problems can only be solved within it).
2. The solutions are sufficiently open-ended to suggest a large number of interesting related problems.

The PDL system and its underlying picture processing model satisfy these criteria. The preceding chapters contain discussions of the system's advantages, limitations, and possible extensions. These are now completed and summarized under two headings--summary of features (criterion 1), and future work (corresponding to criterion 2).

8.1 SUMMARY OF FEATURES

The PDL picture description language is evaluated below in terms of the requirements which were enumerated at the beginning of Chapter 3:

1. <u>Descriptive Range</u>

The examples of Chapter 4 and the application of Chapter 7 indicate that many different classes of pictures can be described by PDL in a meaningful way both to humans and machines.

2. Completeness of a Description

A PDL description contains the structure and meaning of the picture. Imposed semantics must be added for complete descriptive capability.

3. A Simple and Natural Formalism

The PDL description scheme is certainly simple. Whether it is natural or not is mainly a subjective judgement on the part of the reader.

4. Generative Descriptions

The abstracted version of the picture produced during the parse is actually a generation based on the picture PDL description. Any PDL description contains in a usable form the information necessary for generation.

5. Direct Use of the Language for Analysis and Generation

The PDL description explicitly directs the parse of pictures. Preliminary work indicates that the generation schemes can employ the description directly in a similar manner.

6. General Algorithms for Analysis and Generation

A general algorithm which applies to any picture which may be described by PDL has been presented. Similar algorithms for generation must await future work.

7. Independence of Digitization

The language is (almost) independent of the digitization mechanism; the latter directly affects only the primitive recognizers. Thus, primitive and hierarchic structural descriptions are generally constant within a given class of pictures regardless, for example, of whether a rectangular or hexagonal grid system with either binary or grey-level codes is used.

8. <u>Applicability to n-dimensional Pictures</u>

PDL structural descriptions are independent of the dimension of the picture. Pictures in n-space for $n > 3$ can be described as well as the normal 2 and 3 dimensional patterns.

The results of this research indicate that the developed picture processing model has several advantageous features when compared with other approaches. These include the generality of both the descriptive and analysis mechanisms, the ability to describe meaningfully a large and interesting class of pictures, the ease of implementation and modification of an analysis system for a particular set of pictures, and the simplification of the basic pattern recognition tasks due to the directed nature of the parser.

8.2 FUTURE WORK

1. <u>The PDL Language and Description Scheme</u>

The PDL language should be extended so that more complex relations among picture components may be expressed; a suggestion in this direction is made in section 4.6. The development of a suitable picture processing language is a desirable first step before the introduction of an imposed semantics; the imposed semantics of a PDL description can then be stated in the picture processing language. At the same time, this could be used to describe the algorithms of the primitive recognizers. Further work on the theoretical properties of the PDL language should include an investigation of methods for manipulating descriptions to prove equivalence and weak equivalence; these results can then be compared with graph matching techniques. A deeper study is warranted on the relationship between picture transformations and their descriptions; transformations to be studied include the affine (matrix) transformation and logical operations on pictures, such as complementation, union and intersection.

2. Parsing and Implementation

An algorithm for converting grammars to PDL standard form is needed; the implemented parser can then be generalized to treat the full language without prior manual grammar conversion. Some study should be made on the usefulness and techniques of embedding the PDL system in a general-purpose programming language such as ALGOL or PL/1; the purpose is to allow picture processing within a larger framework of computations.

3. Primitive Recognition

An interactive graphics system should be written for testing primitive recognizers. This would also provide much insight into blank and don't care primitives, and perhaps yield some more general techniques for their recognition. Primitive recognizers using grey-level digitizing (many intensity levels rather than binary) would extend the applicability of the implemented system.

4. Generation

Generation of pictures may occur in a passive or an active mode, or a combination of these. In all cases, it would be useful to embed PDL in a general-purpose programming language to allow program computation of pictures. An interactive system where generation can be directed by an on-line user requires methods for imposing semantics on the generated pictures; design problems can then be treated in the system. By combining generation and parsing, the computer would be able to participate more effectively in the design process.

5. Data Structures

Most of the analysis time is spent accessing data for primitive recognition; it is expected that primitive generation will consume an equally proportional amount of time. For these reasons, the data structures used at the primitive level are critical, and thus worthy of deeper study. The parsing system at the non-primitive level has a natural data structure determined by the form of PDL expressions; this appears to allow efficient processing.

6. Applications

The extent of usefulness of the PDL system can only be determined by implementing new applications. A number of obvious ones exist:

(a) analysis of spark, streamer, and bubble chamber film, perhaps after three-dimensional reconstruction of the digitizings,

(b) flow chart generation and analysis,

(c) analysis and generation of text including special characters, for example, mathematical notation,

(d) analysis and generation of line drawings, such as electric circuits and bridges,

(e) graph matching and manipulation, and

(f) description of algorithms for parallel processing.

This future work list could be easily extended and expanded. It is clear that there remains a large number of interesting, useful, and challenging problems.

APPENDIX A

PARSING OUTPUT FOR COLLIDING BEAM PICTURES

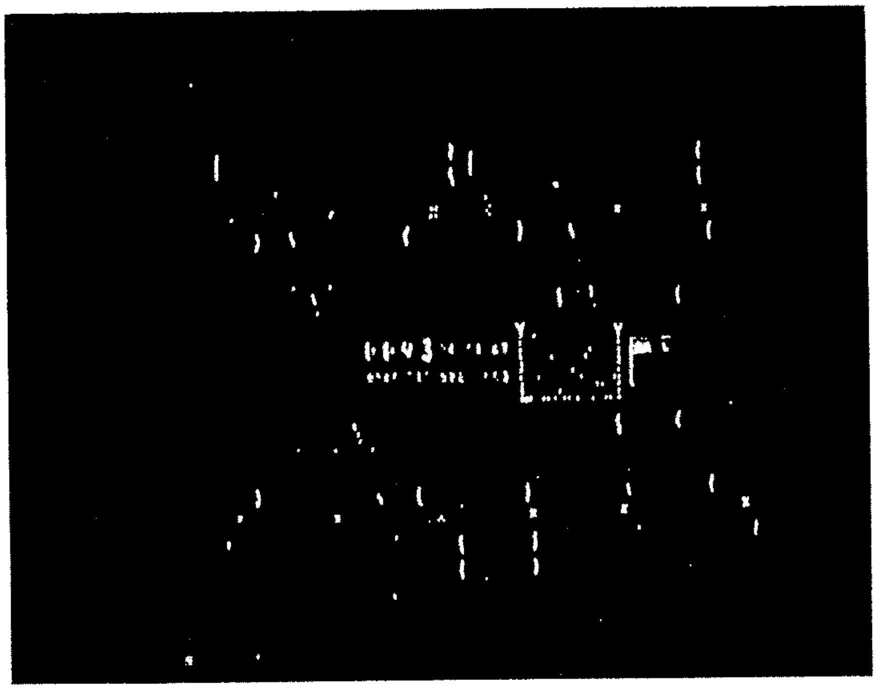
The Appendix contains two photographs of the 2250 display for each frame analyzed by the SPDL system. The first picture is the frame after digitization by the Hummingbird; the second one is the abstracted version of the picture at the completion of the parse. Frame 455 was used to set parameters and debug the grammar and recognizers.

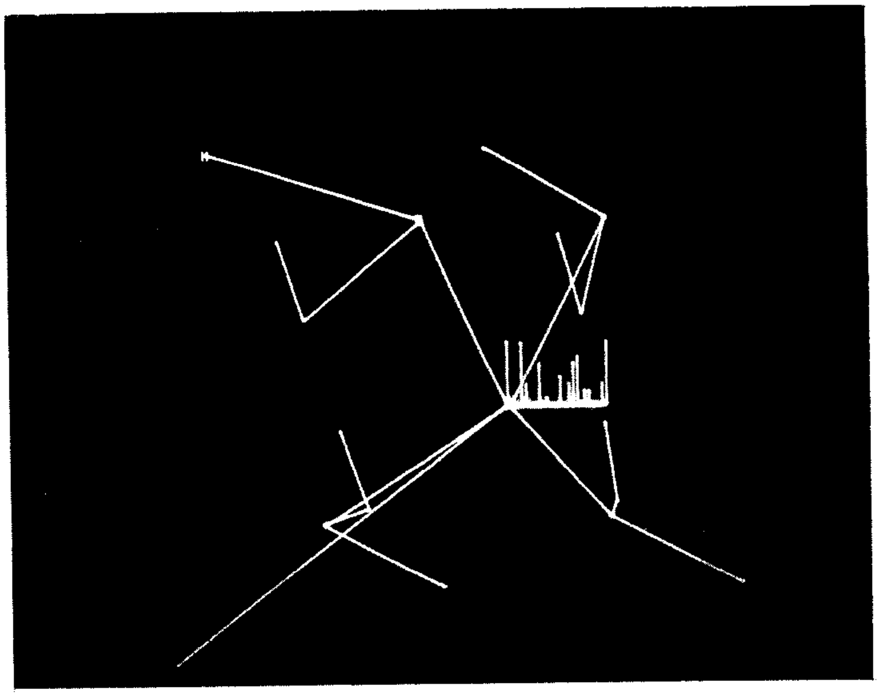
A listing of the parsing tree, natural semantics, and primitive description of frame 355 follows the photographs. The "OTHER VALUES" part of the value list is interpreted:

1. DIGT : the digit represented by the data box code.

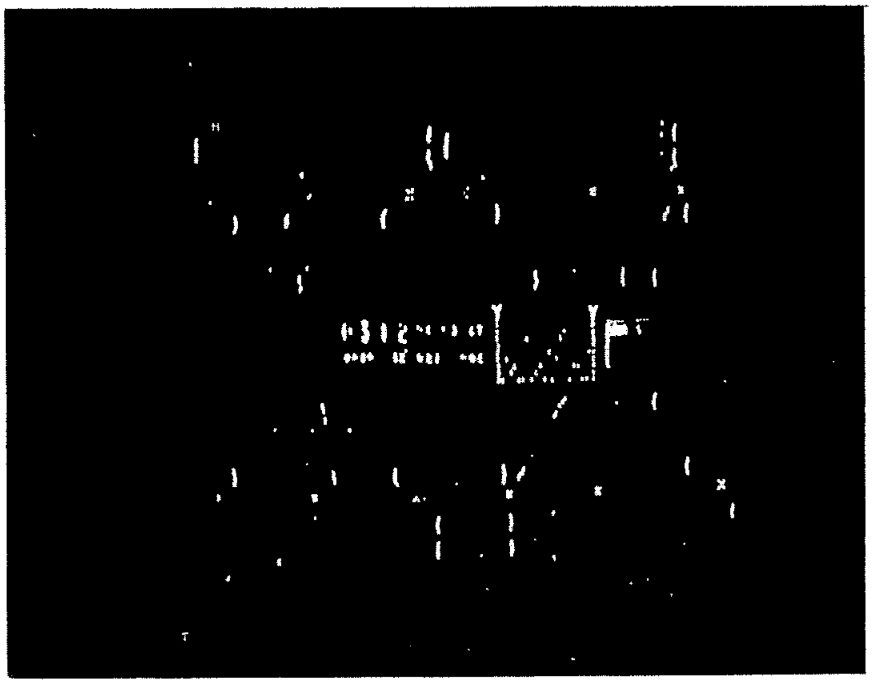
2. EVN$\begin{Bmatrix}\text{T}\\ \text{H}\end{Bmatrix}$: the angle in degrees and the number of sparks. Note that the 2250 display is on a 1024 x 1024 grid, while the data is based on a 4096 x 1024 grid; for display purposes, the x coordinate was divided by four. The output angle is based on the data coordinates.

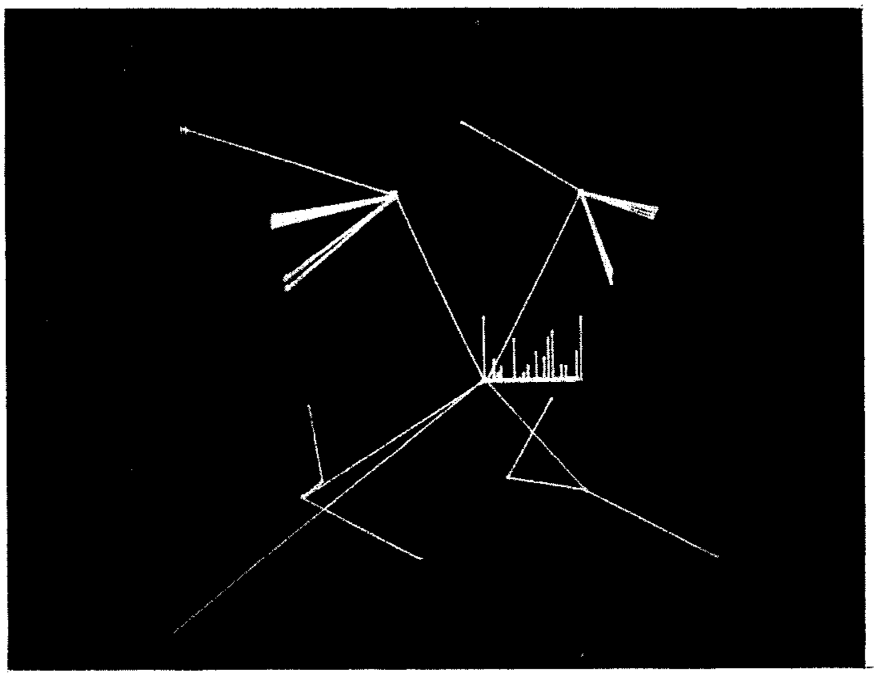
3. SH$\begin{Bmatrix}\text{ST}\\ \text{SB}\\ \text{FT}\\ \text{FB}\end{Bmatrix}$: the number of sparks followed by the coordinates (x, y) of the spark centers.



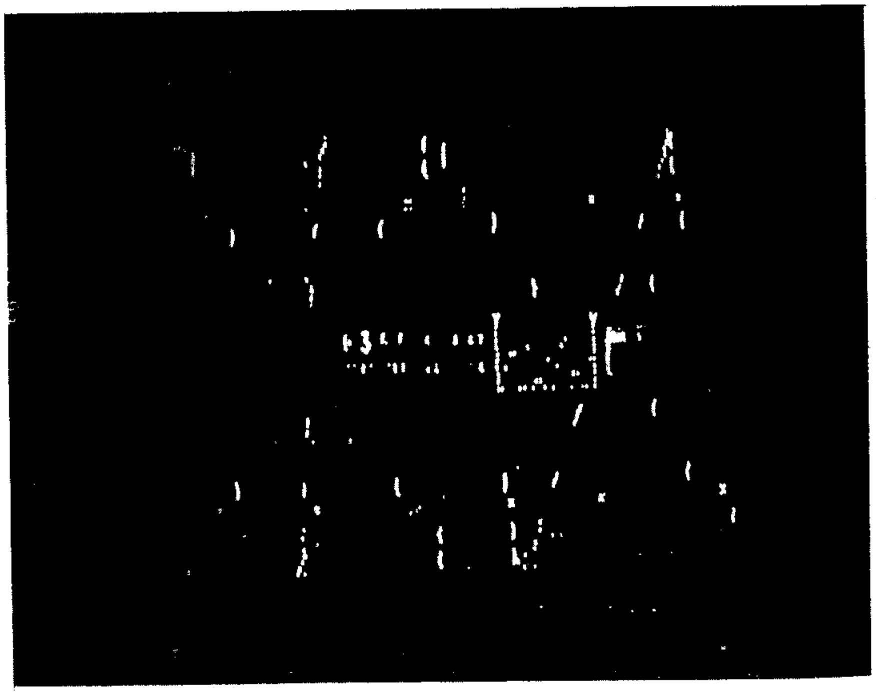


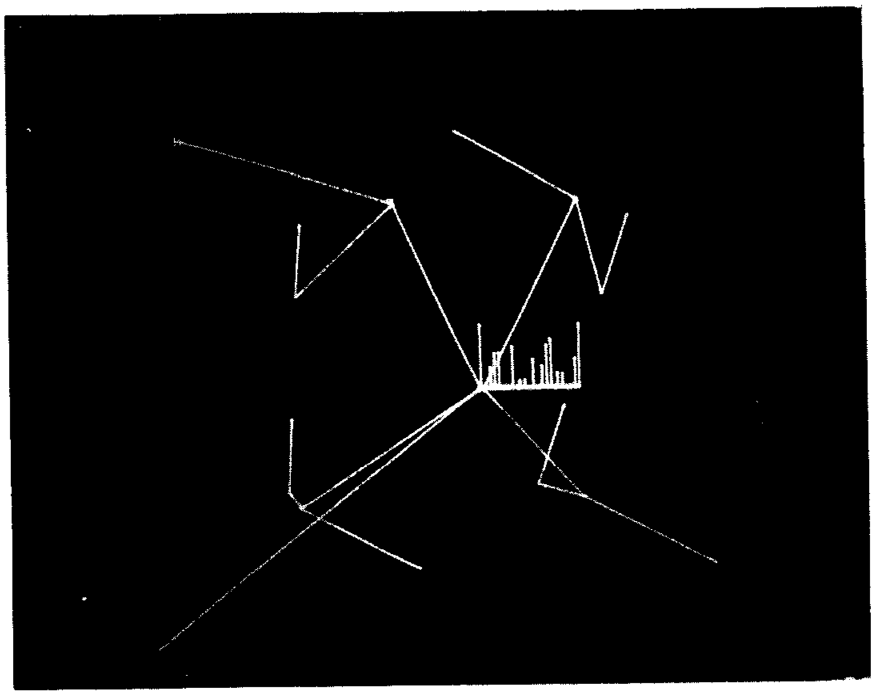
Frame 93



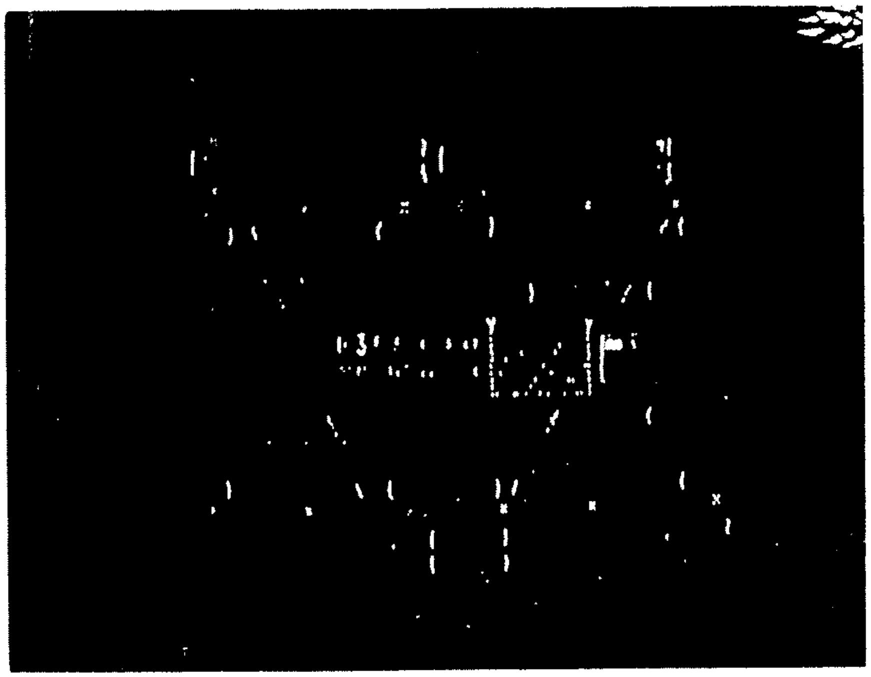


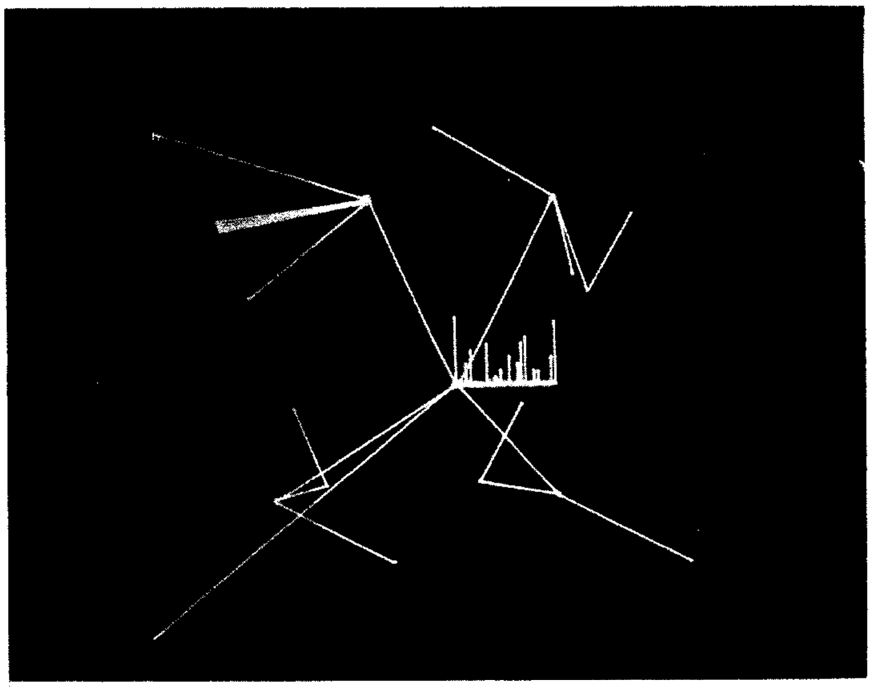
Frame 312



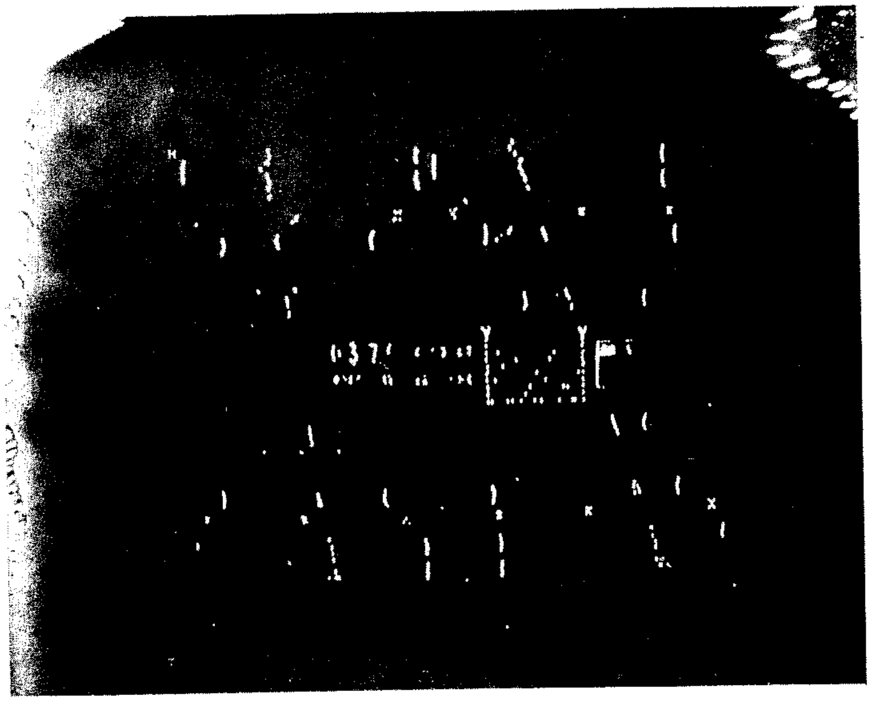


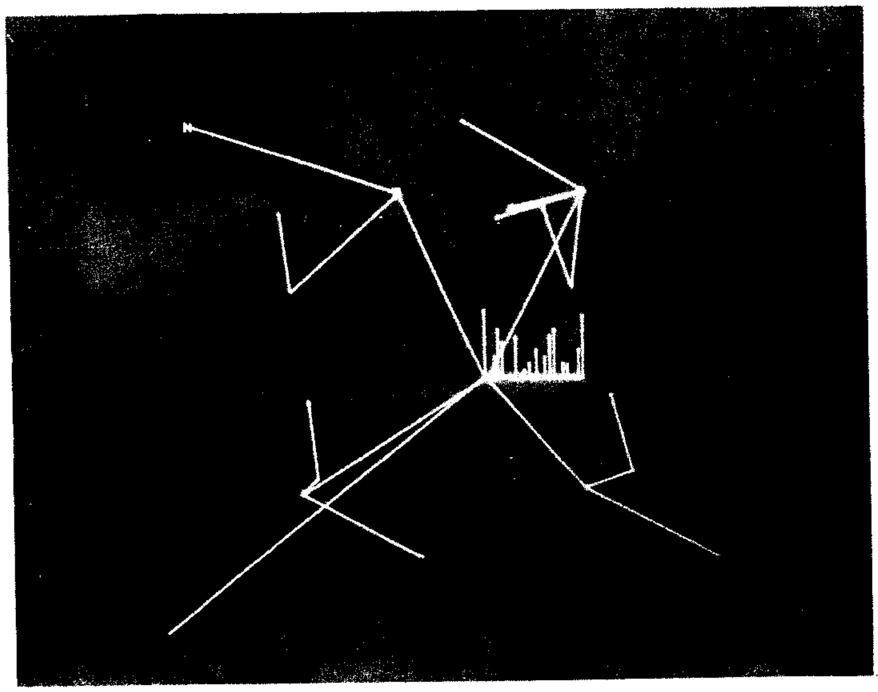
Frame 355



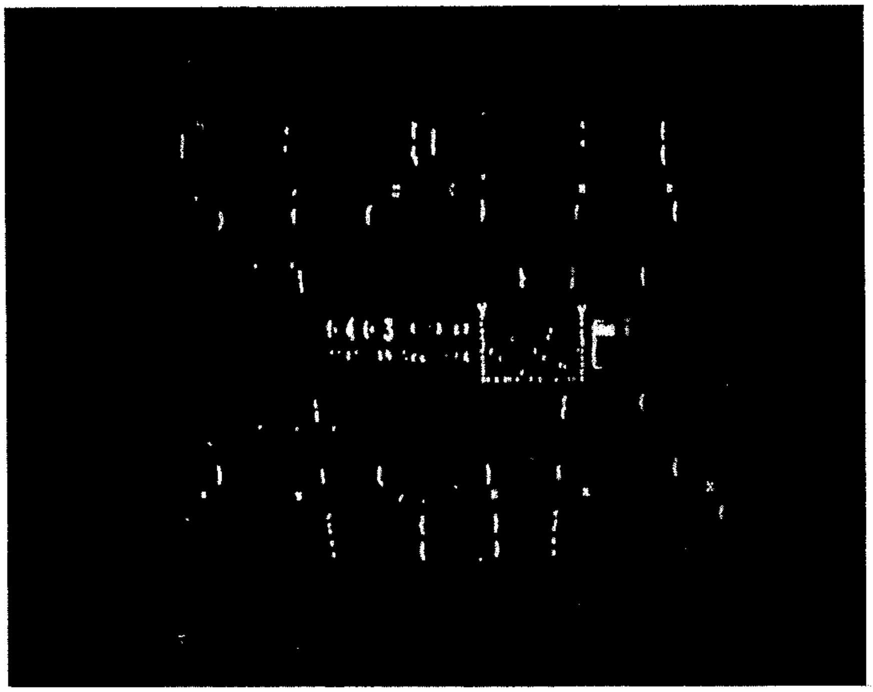


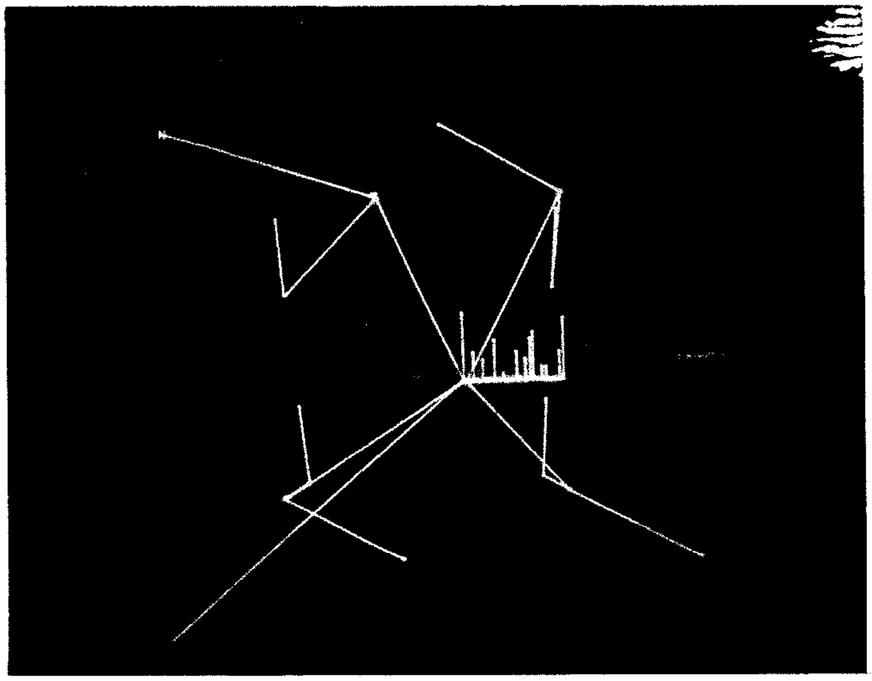
Frame 356



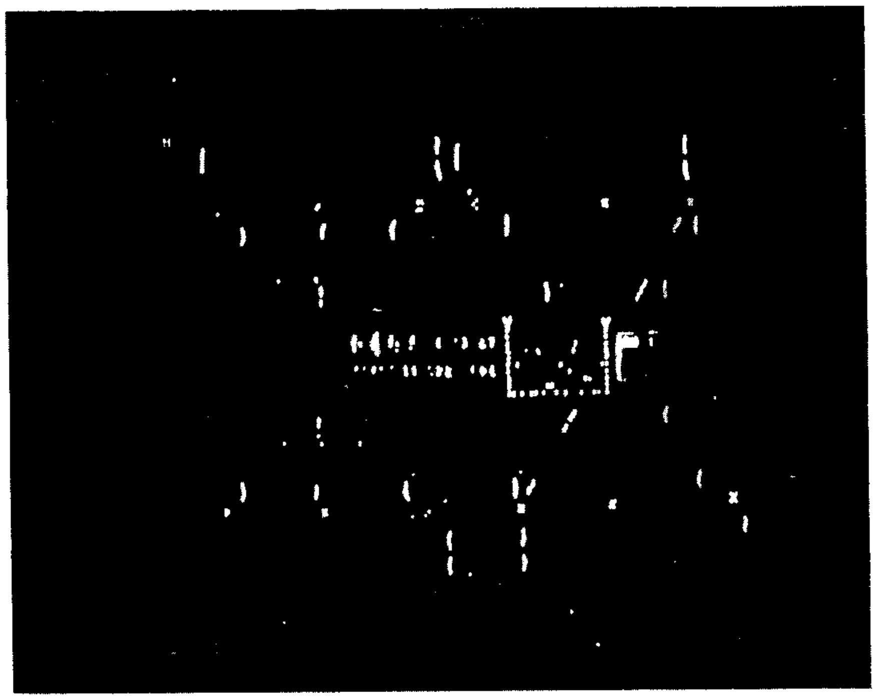


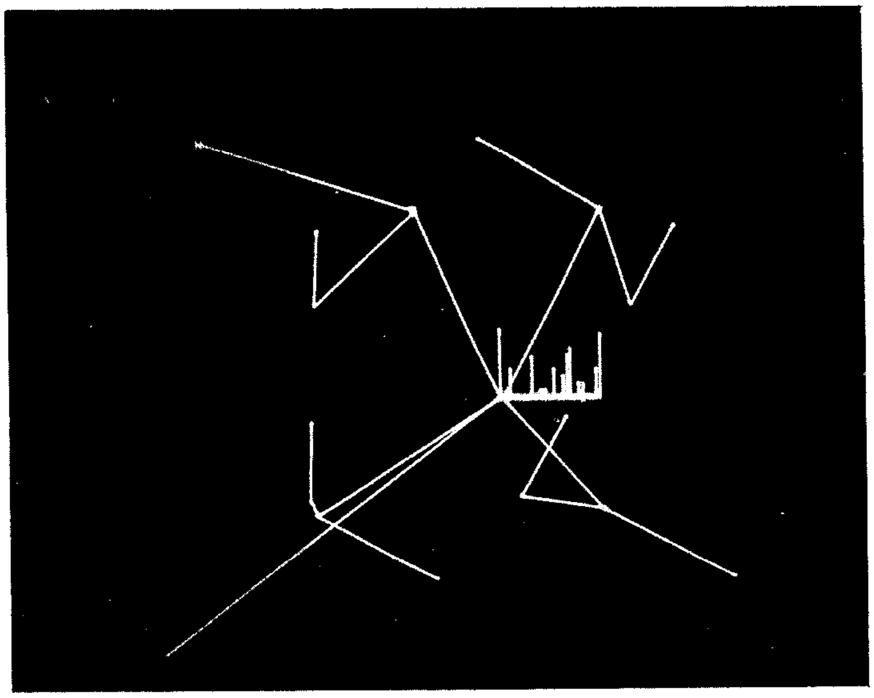
Frame 375



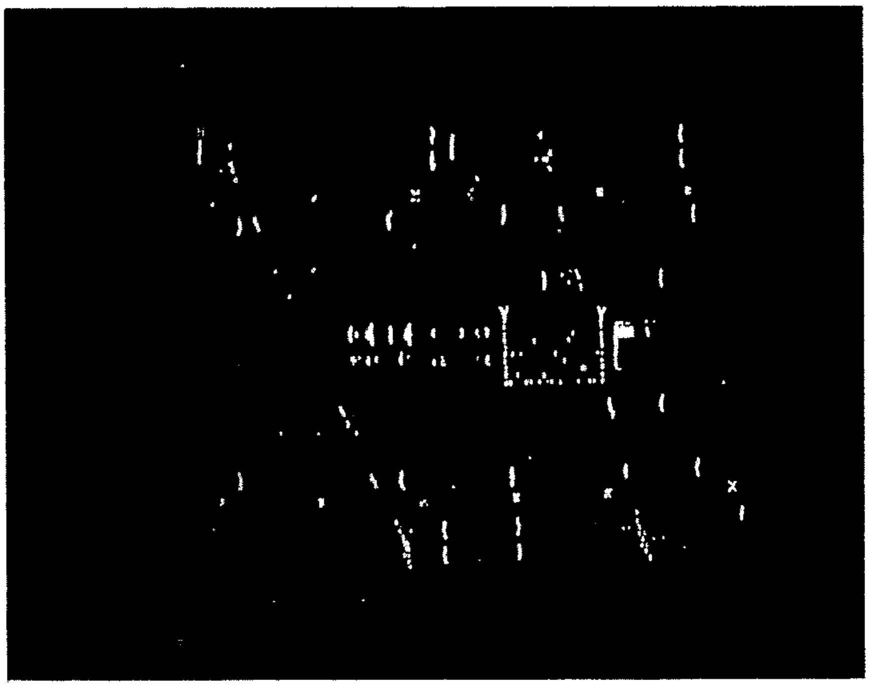


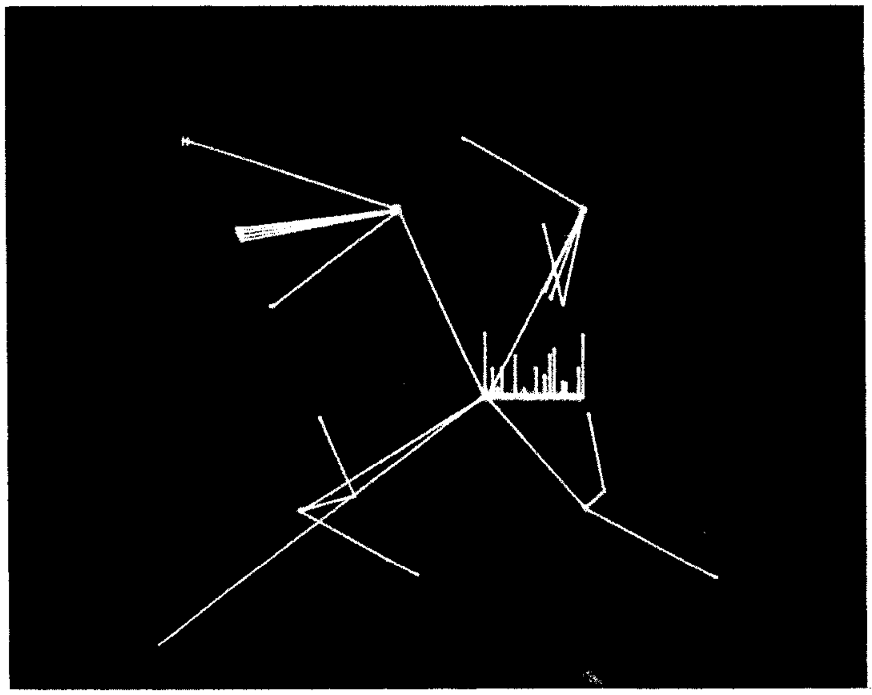
Frame 403



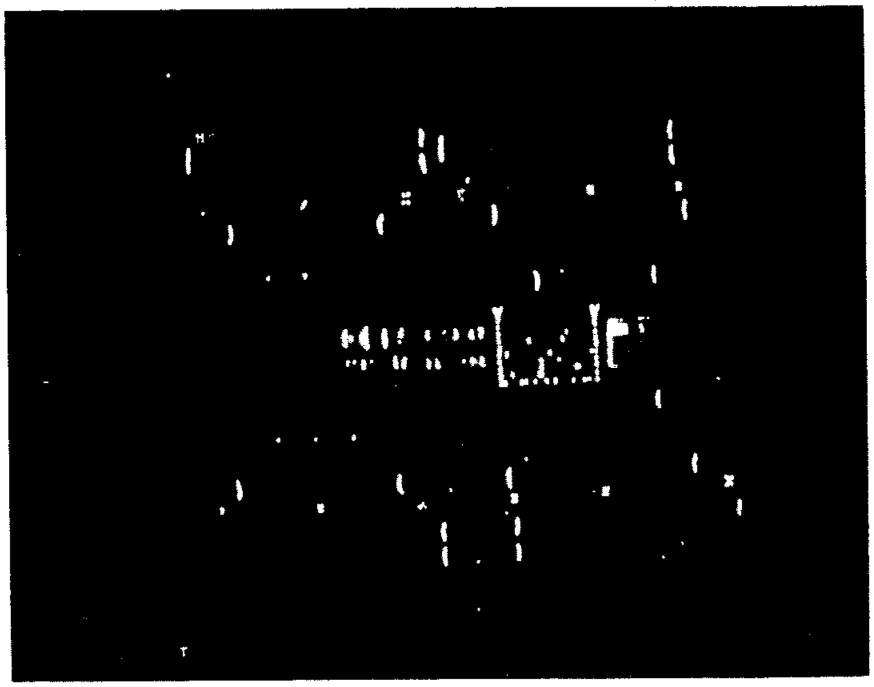


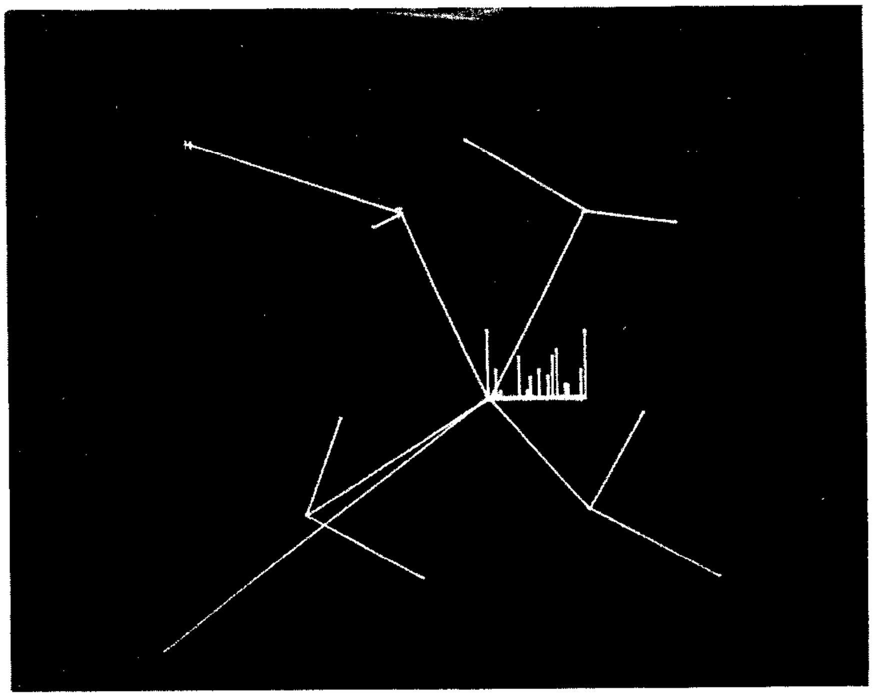
Frame 406



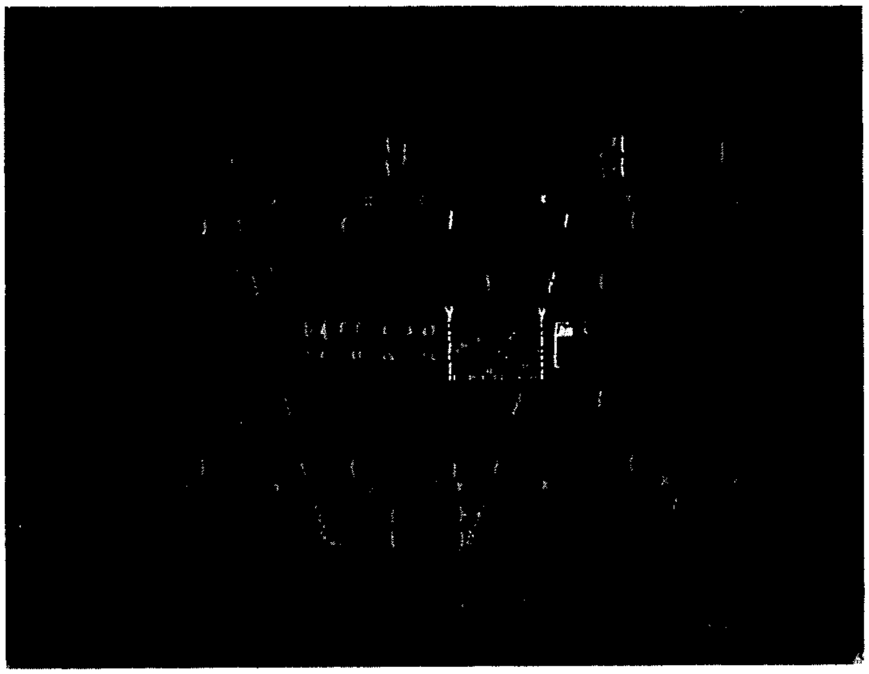


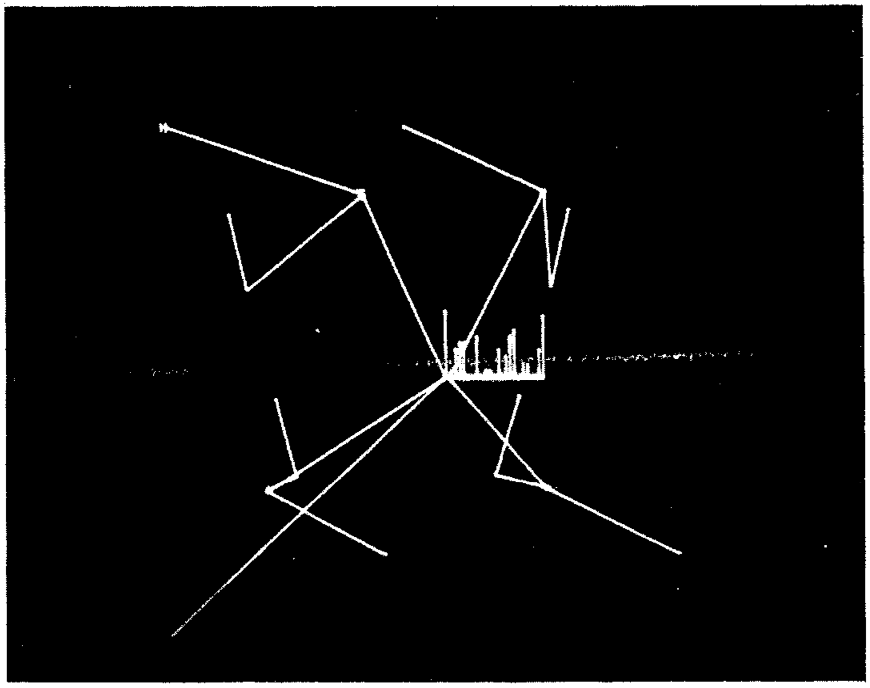
Frame 414





Frame 416





Frame 455

```
PICTURE PARSE WAS SUCCESSFUL..
 NATURAL SEMANTICS OF PICTURE
 PARSING TREE...
     S    GX   ALT   SUP XSUP LSUC RSUC  LOC TLPT HOPT           NAME  OP

     1     1     1     0    1    2    0    1    3  677           CLBM
     2    13     1     1    2    3    4    1    3  677                  +
     3   836     9     2    4    0    0    1   11   14           STRT
     4     9     1     2    3    5  123    1   14  677                  .
     5    17     1     4    4    6    0    1   14  420           CDB
     6    33     1     5    2    7    8    1   14  420                  .
     7   800    17     6    4    0    0    1   19   22           BBND
     8    29     1     6    3    9   10    1   19  420                  +
     9   816    25     8    4    0    0    1   27   30           IDS
    10    25     1     8    3   11  120    1   30  420                  +
    11    37     1    10    4   12    0    1   30  412           DATA
    12    46     1    11    2   13   14    1   30  412                  .
    13   810    33    12    4    0    0    1   35   38           DIGT
    14    42     1    12    3   15   16    1   35  412                  +
    15   816    42    14    4    0    0    1   44   47           IDS
    16    37     1    14    3   17    0    1   47  412           DATA
    17    46     1    16    2   18   19    1   47  412                  .
    18   810    50    17    4    0    0    1   52   55           DIGT
    19    42     1    17    3   20   21    1   52  412                  +
    20   816    59    19    4    0    0    1   61   64           IDS
    21    37     1    19    3   22    0    1   64  412           DATA
    22    46     1    21    2   23   24    1   64  412                  .
    23   810    67    22    4    0    0    1   69   72           DIGT
    24    42     1    22    3   25   26    1   69  412                  +
    25   816    76    24    4    0    0    1   78   81           IDS
    26    37     1    24    3   27    0    1   81  412           DATA
    27    46     1    26    2   28   29    1   81  412                  .
    28   810    84    27    4    0    0    1   86   89           DIGT
    29    42     1    27    3   30   31    1   86  412                  +
    30   816    93    29    4    0    0    1   95   98           IDS
    31    37     1    29    3   32    0    1   98  412           DATA
    32    46     1    31    2   33   34    1   98  412                  .
```

Frame 355 Parse Output

```
33  810  101  32  4   0   0  1  103  106      DIGT
34   42    1  32  3  35  36  1  103  412              +
35  816  110  34  4   0   0  1  112  115      IDS
36   37    1  34  3  37   0  1  115  412      DATA
37   46    1  36  2  38  39  1  115  412
38  810  118  37  4   0   0  1  120  123      DIGT
39   42    1  37  3  40  41  1  120  412              +
40  816  127  39  4   0   0  1  129  132      IDS
41   37    1  39  3  42   0  1  132  412      DATA
42   46    1  41  2  43  44  1  132  412              .
43  810  135  42  4   0   0  1  137  140      DIGT
44   42    1  42  3  45  46  1  137  412              +
45  816  144  44  4   0   0  1  146  149      IDS
46   37    1  44  3  47   0  1  149  412      DATA
47   46    1  46  2  48  49  1  149  412              .
48  810  152  47  4   0   0  1  154  157      DIGT
49   42    1  47  3  50  51  1  154  412              +
50  816  161  49  4   0   0  1  163  166      IDS
51   37    1  49  3  52   0  1  166  412      DATA
52   46    1  51  2  53  54  1  166  412              .
53  810  169  52  4   0   0  1  171  174      DIGT
54   42    1  52  3  55  56  1  171  412              +
55  816  178  54  4   0   0  1  180  183      IDS
56   37    1  54  3  57   0  1  183  412      DATA
57   46    1  56  2  58  59  1  183  412              .
58  810  186  57  4   0   0  1  188  191      DIGT
59   42    1  57  3  60  61  1  188  412              +
60  816  195  59  4   0   0  1  197  200      IDS
61   37    1  59  3  62   0  1  200  412      DATA
62   46    1  61  2  63  64  1  200  412              .
63  810  203  62  4   0   0  1  205  208      DIGT
64   42    1  62  3  65  66  1  205  412              +
65  816  212  64  4   0   0  1  214  217      IDS
66   37    1  64  3  67   0  1  217  412      DATA
67   46    1  66  2  68  69  1  217  412              .
68  810  220  67  4   0   0  1  222  225      DIGT
69   42    1  67  3  70  71  1  222  412              +
```

Frame 355 Parse Output cont.

```
 70  816  229   69   4    0    0    1  231  234         IDS
 71   37    1   69   3   72    0    1  234  412         DATA
 72   46    1   71   2   73   74    1  234  412                 .
 73  810  237   72   4    0    0    1  239  242         DIGT
 74   42    1   72   3   75   76    1  239  412                 +
 75  816  246   74   4    0    0    1  248  251         IDS
 76   37    1   74   3   77    0    1  251  412         DATA
 77   46    1   76   2   78   79    1  251  412                 .
 78  810  254   77   4    0    0    1  256  259         DIGT
 79   42    1   77   3   80   81    1  256  412                 +
 80  816  263   79   4    0    0    1  265  268         IDS
 81   37    1   79   3   82    0    1  268  412         DATA
 82   46    1   81   2   83   84    1  268  412                 .
 83  810  271   82   4    0    0    1  273  276         DIGT
 84   42    1   82   3   85   86    1  273  412                 +
 85  816  280   84   4    0    0    1  282  285         IDS
 86   37    1   84   3   87    0    1  285  412         DATA
 87   46    1   86   2   88   89    1  285  412                 .
 88  810  288   87   4    0    0    1  290  293         DIGT
 89   42    1   87   3   90   91    1  290  412                 +
 90  816  297   89   4    0    0    1  299  302         IDS
 91   37    1   89   3   92    0    1  302  412         DATA
 92   46    1   91   2   93   94    1  302  412                 .
 93  810  305   92   4    0    0    1  307  310         DIGT
 94   42    1   92   3   95   96    1  307  412                 +
 95  816  314   94   4    0    0    1  316  319         IDS
 96   37    1   94   3   97    0    1  319  412         DATA
 97   46    1   96   2   98   99    1  319  412                 .
 98  810  322   97   4    0    0    1  324  327         DIGT
 99   42    1   97   3  100  101    1  324  412                 +
100  816  331   99   4    0    0    1  333  336         IDS
101   37    1   99   3  102    0    1  336  412         DATA
102   46    1  101   2  103  104    1  336  412                 .
```

Frame 355 Parse Output cont.

```
103   810   339   102    4    0    0     1   341   344           DIGT
104    42     1   102    3  105  106     1   341   412                  +
105   816   348   104    4    0    0     1   350   353           IDS
106    37     1   104    3  107    0     1   353   412           DATA
107    46     1   106    2  108  109     1   353   412                  .
108   810   356   107    4    0    0     1   358   361           DIGT
109    42     1   107    3  110  111     1   358   412                  +
110   816   365   109    4    0    0     1   367   370           IDS
111    37     1   109    3  112    0     1   370   412           DATA
112    46     1   111    2  113  114     1   370   412                  .
113   810   373   112    4    0    0     1   375   378           DIGT
114    42     1   112    3  115  116     1   375   412                  +
115   816   382   114    4    0    0     1   384   387           IDS
116    37     2   114    3  117    0     1   387   412           DATA
117    50     1   116    2  118  119     1   387   412                  .
118   810   390   117    4    0    0     1   392   395           DIGT
119   826   407   117    3  120  121     1   409   412           NULL
120    21     1    10    3  121  122     1   412   420                  +
121   816   399   120    4  122    0     1   401   404           IDS
122   800   415   120    3  123    0     1   417   420           BBND
123     5     1     4    3  124  162     1    14   677                  .
124    70     1   123    4  125    0     1    14   539           FRVW
125    82     1   124    2  126  144     1    14   539                  .
126    74     1   125    4  127  128     1    14   470                  +
127   838   423   126    4    0    0     1   425   428           XFB
128   110     1   126    3  129    0     1   428   470           CFB
129   114     1   128    2  130  143     1   428   470                  .
130   127     2   129    4  131    0     1   428   462           FB46
131   132     1   130    2  132  141     1   428   462                  .
132   136     1   131    4  133    0     1   428   454           L04
133   145     1   132    2  134  138     1   428   454                  .
134   141     1   133    4  135  136     1   428   444                  +
135   804   431   134    4    0    0     1   433   436           B4GL
```

Frame 355 Parse Output cont.

```
136  154    1  134   3  137    0   1  441  444          CLO
137  814  439  136   2    0    0   1  441  444          EVNT
138  149    1  133   3  139    0   1  451  454          LO4D
139  136    2  138   2  140    0   1  451  454          LO4
140  826  449  139   2  141    0   1  451  454          NULL
141  163    2  131   3  142    0   1  459  462          LO6
142  826  457  141   2  143    0   1  459  462          NULL
143  828  465  129   3  144    0   1  467  470          SHFB
144   78    1  125   3  145  146   1   14  539               +
145  840  492  144   4    0    0   1  494  497          XFT
146   94    1  144   3  147    0   1  497  539          CFT
147   98    1  146   2  148  161   1  497  539               .
148  190    2  147   4  149    0   1  497  531          FT46
149  195    1  148   2  150  159   1  497  531               .
150  222    1  149   4  151    0   1  497  523          HI6
151  231    1  150   2  152  156   1  497  523               .
152  227    1  151   4  153  154   1  497  513               +
153  806  500  152   4    0    0   1  502  505          B6GH
154  217    1  152   3  155    0   1  510  513          CHI
155  812  508  154   2  156    0   1  510  513          EVNH
156  235    1  151   3  157    0   1  520  523          HI6D
157  222    2  156   2  158    0   1  520  523          HI6
158  826  518  157   2  159    0   1  520  523          NULL
159  199    2  149   3  160    0   1  528  531          HI4
160  826  526  159   2  161    0   1  528  531          NULL
161  830  534  147   3  162    0   1  536  539          SHFT
162   54    1  123   3  163    0   1   14  677          SDVW
163   66    1  162   2  164  182   1   14  677               .
164   58    1  163   4  165  166   1   14  606               +
165  842  559  164   4  166    0   1  561  564          XSB
166  102    1  164   3  167    0   1  564  606          CSB
167  106    1  166   2  168  181   1  564  606               .
168  118    2  167   4  169    0   1  564  598          SB46
```

Frame 355 Parse Output cont.

```
169  123    1  168   2  170  179   1  564  598            .
170  136    1  169   4  171    0   1  564  590      LO4
171  145    1  170   2  172  176   1  564  590            .
172  141    1  171   4  173  174   1  564  580            +
173  804  567  172   4  174    0   1  569  572      B4GL
174  154    1  172   3  175    0   1  577  580      CLO
175  814  575  174   2  176  181   1  577  580      EVNT
176  149    1  171   3  177    0   1  587  590      LO4D
177  136    2  176   2  178    0   1  587  590      LO4
178  826  585  177   2  179    0   1  587  590      NULL
179  163    2  169   3  180    0   1  595  598      LO6
180  826  593  179   2  181    0   1  595  598      NULL
181  832  601  167   3  182    0   1  603  606      SHSB
182   62    1  163   3  183  184   1   14  677            +
183  844  630  182   4  184  189   1  632  635      XST
184   86    1  182   3  185    0   1  635  677      CST
185   90    1  184   2  186  199   1  635  677            .
186  181    2  185   4  187    0   1  635  669      ST46
187  186    1  186   2  188  197   1  635  669            .
188  222    1  187   4  189    0   1  635  661      HI6
189  231    1  188   2  190  194   1  635  661            .
190  227    1  189   4  191  192   1  635  651            +
191  806  638  190   4  192    0   1  640  643      B6GH
192  217    1  190   3  193    0   1  648  651      CHI
193  812  646  192   2  194  198   1  648  651      EVNH
194  235    1  189   3  195    0   1  658  661      HI6D
195  222    2  194   2  196    0   1  658  661      HI6
196  826  656  195   2  197    0   1  658  661      NULL
197  199    2  187   3  198    0   1  666  669      HI4
198  826  664  197   2  199    0   1  666  669      NULL
199  834  672  185   3  200    0   1  674  677      SHST
```

Frame 355 Parse Output cont.

```
PRIMITIVE VALUE LIST...

NAME    XT    YT    ZT    XH    YH    ZH   OTHER VALUES

STRT     1     1     0  2308   439     0
BBND  2308   439     0  2308   550     0
IDS   2308   439     0  2338   439     0
DIGT  2338   439     0  2338   439     0     0
IDS   2338   439     0  2368   439     0
DIGT  2368   439     0  2368   475     0     3
IDS   2368   439     0  2397   439     0
DIGT  2397   439     0  2397   498     0     5
IDS   2397   439     0  2427   439     0
DIGT  2427   439     0  2427   499     0     5
IDS   2427   439     0  2455   439     0
DIGT  2455   439     0  2455   439     0     0
IDS   2455   439     0  2484   439     0
DIGT  2484   439     0  2484   440     0     0
IDS   2484   439     0  2515   439     0
DIGT  2515   439     0  2515   510     0     6
IDS   2515   439     0  2543   439     0
DIGT  2543   439     0  2543   440     0     0
IDS   2543   439     0  2571   439     0
DIGT  2571   439     0  2571   451     0     1
IDS   2571   439     0  2600   439     0
DIGT  2600   439     0  2600   452     0     1
IDS   2600   439     0  2629   439     0
DIGT  2629   439     0  2629   440     0     0
IDS   2629   439     0  2658   439     0
DIGT  2658   439     0  2658   487     0     4
IDS   2658   439     0  2685   439     0
DIGT  2685   439     0  2685   439     0     0
IDS   2685   439     0  2715   439     0
DIGT  2715   439     0  2715   476     0     3
IDS   2715   439     0  2745   439     0
DIGT  2745   439     0  2745   511     0     6
IDS   2745   439     0  2772   439     0
```

Frame 355 Parse Output cont.

```
DIGT 2772   439     0 2772   522     0     7

IDS  2772   439     0 2800   439     0

DIGT 2800   439     0 2800   439     0     0

IDS  2800   439     0 2827   439     0

DIGT 2827   439     0 2827   463     0     2

IDS  2827   439     0 2856   439     0

DIGT 2856   439     0 2856   462     0     2

IDS  2856   439     0 2884   439     0

DIGT 2884   439     0 2884   440     0     0

IDS  2884   439     0 2912   439     0

DIGT 2912   439     0 2912   440     0     0

IDS  2912   439     0 2941   439     0

DIGT 2941   439     0 2941   487     0     4

IDS  2941   439     0 2967   439     0

NULL 2941   439     0 2941   439     0

BBND 2967   439     0 2969   547     0

XFB  2308   439     0 2986   247     0

B4GL 2986   247     0 2667   271     0

EVNT 2667   271     0 2867   407     0    34     10

NULL 2986   247     0 2986   247     0

NULL 2986   247     0 2986   247     0

SHFB 2986   247     0 3889   127     0      9 2445   129 2513   131 2401   140 2450   151 2516   163 2519   172 2646
        184 2554  194 2562   209

XFT  2308   439     0 2978   760     0

B6GH 2978   760     0 3143   597     0

EVNH 3143   597     0 3333   731     0    34      9

NULL 2978   760     0 2973   760     0

NULL 2978   760     0 2978   760     0

SHFT 2978   760     0 2161   883     0      8 3436   797 3453   812 3472   824 3492   833 3487   843 3501   855 3509
        866 3509  876

XSB  2308   439     0 1033   242     0

B4GL 1033   242     0  944   268     0

EVNT  944   268     0  984   395     0    73      8

NULL 1033   242     0 1033   242     0
```

Frame 355 Parse Output cont.

```
NULL 1033  242     0 1033  242     0

SHSB 1033  242     0 1841  130     0   10  920  131  875  131  890  141  905  150  919  161  933  161  919
      171 1009   181  913  192   925  207

XST  2308  439     0 1727  760     0

B6GH 1727  760     0 1047  605     0

EVNH 1047  605     0 1095  728     0   71    7

NULL 1727  760     0 1727  760     0

NULL 1727  760     0 1727  760     0

SHST 1727  760     0  271  878     0    9 1124  799 1128  814 1141  825 1046  832 1044  835 1140  845 1160
      855 1161   860 1166  867
```

Frame 355 Parse Output cont.

REFERENCES

All GSG memos (marked with an asterisk below) are internal working papers of the Graphics Study Group at the Stanford Linear Accelerator Center, and may be obtained from their respective authors at the following address:

Stanford Linear Accelerator Center
P. O. Box 4349
Stanford, California 94305

Adler, B., Fernbach, S., and Rotenberg, M. [1966]. Methods in Computational Physics, Alt, F. (Ed.). Volume 5, Academic Press, New York.

Alt, F. [1962]. Digital pattern recognition by moments. J. ACM 9, 2 (April), 240-258.

Anderson, R. [1967]. Syntax-directed recognition of hand-printed two-dimensional mathematics. Proceedings of the ACM Symposium on Interactive Systems for Experimental Applied Mathematics (to be published).

Barber, W., Gittelman, B., O'Neill, G., and Richter, B. [1966]. Test of quantum electrodynamics by electron-electron scattering. Physical Review Letters 16, 24 (June), 1127-1130.

Barber, W., Richter B., Gittelman, B., and O'Neill, G. [1965]. Wide angle electron-electron scattering on the Princeton-Stanford storage rings. SLAC-DOC-73, Stanford Linear Accelerator Center, Stanford, California, (September). Also presented at Oxford International Conference on Elementary Particles.

BCS [1967]. Character Recognition, British Computer Society, London.

Bledsoe, W., and Browning, J. [1959]. Pattern recognition and reading by machine. Proceedings of The 1959 Eastern Joint Computer Conference, 225-232.

Böhm, C., and Jacopini, G. [1966]. Flow diagrams, Turing machines, and languages with only two formation rules. Comm. ACM 9, 5 (May), 366-371.

Bomba, J. [1959]. Alpha-numeric character recognition using local operations. Proceedings of The 1959 Eastern Joint Computer Conference, (December), 218-224.

Breeding, K. [1965]. Grammar for a pattern description language. Report No. 177, Department of Computer Science, University of Illinois (May) (M.S. thesis).

*Brown, J. [1967]. Preliminary study of calibration stability of "Hummingbird" film digitizer. GSG Memo 5, Computation Group, Stanford Linear Accelerator Center, Stanford, California (April).

Brown, J. [1967]. Private communication.

Carlbom, J. [1967]. An algorithm for transferring a PDL expression into a primitive connection matrix. Computation Group, Stanford Linear Accelerator Center, Stanford, California (unpublished).

Cheatham, T., and Sattley, K. [1964]. Syntax directed compiling. Proceedings of the AFIPS Spring Joint Computer Conference, Spartan Books, Inc., Washington, D. C., 31-57.

Chomsky, N. [1957]. Syntactic Structures. Mouton and Co., London.

Chomsky, N. [1959]. On certain formal properties of grammars. Information and Control 2, 137-167.

Chomsky, N. [1965]. Aspects of the Theory of Syntax. M.I.T. Press, Cambridge.

Chow, C. [1957]. An optimum character recognition system using decision functions. IRE Transactions on Electronic Computers EC-6, 4 (December), 247-254.

Clark, R. and Miller, W. [1966]. Computer based data analysis systems at Argonne. In Adler et al. [1966], 47-98.

Clowes, M. [1967a]. Perception, picture processing and computers. Machine Intelligence 1, Collins, N., and Michie, D. (Ed.), Oliver and Boyd, London, 181-197.

Clowes, M. [1967b]. A generative picture grammar. Seminar paper no. 6, Computing Research Section, Commonwealth Scientific and Industrial Research Organization, Australia (April).

Dickens, C. [1967a]. Systems for the Hummingbird at SLAC. Proceedings of the 1967 International Conference in Programming for Flying Spot Devices, Powell, B. W., and Seyboth, P. (Ed.), Max Planck Institut für Physik und Astrophysik, Munich (January), 62-72. Also published as SLAC-PUB-255, Stanford Linear Accelerator Center, Stanford, California.

Dickens, C. [1967b]. Private communication.

Dinneen, G. [1955]. Programming pattern recognition. Proceedings of the 1955 Western Joint Computer Conference, Institute of Radio Engineers, 94-100.

Eden, M. [1961]. On the formalization of handwriting. Proceedings of Symposia in Applied Mathematics, American Mathematical Society 12, 83-88.

Eden, M. [1962]. Handwriting and pattern recognition. IRE Transactions on Information Theory IT-8, 2, 160-166.

Feder, J. [1966]. The linguistic approach to pattern analysis--a literature survey. Technical Report 400-133, Department of Electrical Engineering, New York University (February).

Feigenbaum, E., and Feldman, J. (Ed.)[1963]. Computers and Thought. McGraw-Hill, New York.

Feldman, J. [1966]. A formal semantics for computer languages and its application in a compiler-compiler. Comm. ACM 9, 1 (January), 3-9.

Feldman, J., and Gries, D. [1967]. Translator writing systems. Report No. CS 69, Computer Science Department, Stanford University (June).

Floyd, R. [1964]. The syntax of programming languages--a survey. IEEE Transactions on Electronic Computers EC-13, 4 (August), 346-353.

Ford, K. [1963]. The World of Elementary Particles. Blaisdell Publishing Company, New York.

Freeman, H. [1961]. On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computers EC-10, 2 (June), 260-268.

*George, J. [1967]. Picture Generation based on the picture calculus. GSG Memo 50, Computation Group, Stanford Linear Accelerator Center, Stanford, California (December).

Ginsburg, S. [1966]. The Mathematical Theory of Context-Free Languages. McGraw-Hill, New York.

Gittelman, B. [1967]. Scanning instructions for the colliding beam film. Stanford Linear Accelerator Center, Stanford, California (unpublished).

Grimsdale, R., Sumner, F., Tunis, C., and Kilburn, T. [1958]. A system for the automatic recognition of patterns. Paper No. 2792 M, The Institution of Electrical Engineering (December), 210-221.

Guzman, A. [1967]. Some aspects of pattern recognition by computer. MAC-TR-37, Project MAC, Massachusetts Institute of Technology, (February) (M.S. thesis).

Hu, M. [1962]. Visual pattern recognition by moment invariants. IEEE Transactions on Information Theory 8, 2 (February) 179-187.

IBM [1965]. IBM operating system/360 concepts and facilities. Form C28-6535-0, IBM Corporation, White Plains, New York.

IBM [1966]. IBM system/360 FORTRAN IV language, Form C28-6514-4, IBM Corporation, White Plains, New York.

IBM [1967]. IBM system/360 system summary. Form A22-6810-8, IBM Corporation, White Plains, New York.

Irons, E. [1961]. A syntax directed compiler for ALGOL 60. Comm. ACM 4, 1 (January), 51-55.

Kirsch, R. [1964]. Computer interpretation of English text and picture patterns. IEEE Transactions on Electronic Computers EC-13, 4 (August), 363-376.

Knuth, D. [1963]. Computer-drawn flowcharts. Comm. ACM 6, 9 (September), 555-563.

Kuhn, T. S. [1962]. <u>The Structure of Scientific Revolutions</u>. The University of Chicago Press, Chicago.

Leavenworth, B. [1964]. FORTRAN IV as a syntax language. Comm. ACM 7, 2 (February), 72-80.

Ledley, R. [1962]. <u>Programming and Utilizing Digital Computers</u>. McGraw-Hill, New York, Chapter 8.

Ledley, R., Rotolo, L., Golab, T., Jacobsen, J., Ginsberg, M., and Wilson, J. [1965]. FIDAC: film input to digital automatic computer and associated syntax-directed pattern recognition programming system. In Tippett et al. [1965], Chapter 33.

Lipkin, L., Watt, W., and Kirsch, R. [1966]. The analysis, synthesis, and description of biological images. <u>Annals of the New York Academy of Sciences</u> 128, 3 (January), 984-1012.

Marill, T., and Green, D. M. [1960]. Statistical recognition functions and the design of pattern recognizers. IRE Transactions on Electronic Computers. EC-9, 4 (December), 472-477.

Marill, T., Hartley, A., Evans, T., Bloom, B., Park, D., Hart, T., and Dailey, D. [1963]. CYCLOPS-1: a second-generation recognition system. <u>Proceedings of the Fall Joint Computer Conference</u>, Spartan Books, Washington, D. C., 27-33.

Marr, R., and Rabinowitz, G. [1966]. A software approach to the automatic scanning of digitized bubble chamber photographs. In Adler et al. [1966], 213-258.

McCarthy, J. [1963]. A basis for a mathematical theory of computation. <u>Computer Programming and Formal Systems</u>, Braffort, P. and Hirschberg, D. (Ed.), North-Holland Publishing Company, Amsterdam, 33-70.

*McGee, W. [1966]. A simple moment invariant. GSG Memo 8, Computation Group, Stanford Linear Accelerator Center, Stanford, California (June).

*Miller, W. [1966]. Globs--soft and hard. GSG Memo 6, Computation Group, Stanford Linear Accelerator Center, Stanford, California (April).

Miller, W. [1967b]. Private communication.

*Miller, W. and Shaw, A. [1967a]. A picture calculus. GSG Memo 40, Computation Group, Stanford Linear Accelerator Center, Stanford, California (June).

Miller, W. and Shaw, A. [1967b]. A picture calculus. Emerging Concepts in Graphics, University of Illinois (November) (to be published).

Miller, W. and Van der Lans, J. [1967]. System design for CRT film scanning and measuring. SLAC-PUB-300, Stanford Linear Accelerator Center, Stanford, California (April). Also presented at the Eighth National Symposium of the Society for Information Display.

Minsky, M. [1961]. A selected descriptor-index bibliography to the literature on artificial intelligence. IRE Transactions on Human Factors in Electronics (March), 30-55. Also in Feigenbaum and Feldman [1963].

Moorhead, W., and Powell, B. (Ed.) [1965]. Programming for Flying Spot Devices. European Organization for Nuclear Research, CERN 65-11, Geneva (March).

Narasimhan, R. [1962]. A linguistic approach to pattern recognition. Report No. 21, Digital Computer Laboratory, University of Illinois (July).

Narasimhan, R. [1963a]. A programming system for scanning digitized bubble-chamber negatives. Report No. 139, Digital Computer Laboratory, University of Illinois (June).

Narasimhan, R. [1963b]. Syntactic descriptions of pictures and gestalt phenomena of visual perception. Report No. 142, Digital Computer Laboratory, University of Illinois (July).

Narasimhan, R. [1964]. Labeling schemata and syntactic description of pictures. Information and Control 7, 151-179.

Narasimhan, R. [1966]. Syntax-directed interpretation of classes of pictures. Comm. ACM 9, 3 (March), 166-173.

Naur, P. (Ed.) [1963]. Revised report on the algorithmic language ALGOL 60. Comm. ACM 6, 1 (January), 1-17.

Nilsson, N. [1965]. Learning Machines. McGraw-Hill, New York.

Noyelle, Y. [1967]. Implementation on the PDL-1 of a subset of the picture calculus. Term project for CS 260, Computer Science Department, Stanford University (Spring Quarter) (unpublished).

Pless, J., Rosenson, L., Bastien, P., Wadsworth, B., Watts, T., Yamamoto, R., Alston, M., Rosenfeld, A., Solmitz, F., and Taft, H. [1965]. A precision encoding and pattern recognition system (PEPR). Paper submitted to the 1964 International Conference on High Energy Physics at Dubna.

Randall, B., and Russell, L. [1964]. ALGOL 60 Implementation. Academic Press, London.

Reynolds, J. [1965]. An introduction to the COGENT programming system. Proceedings of the 20th National ACM Conference, 422-436.

Roberts, L. G. [1963]. Machine perception of three-dimensional solids. Technical Report No. 315, Lincoln Laboratory, Massachusetts Institute of Technology (May).

Rosen, C., and Nilsson, J. (Ed.) [1966]. Application of intelligent automata to reconnaissance. AF 30(602)-4147, Stanford Research Institute, Menlo Park, California (November).

Rosenfeld, A. and Pfaltz, J. [1966]. Sequential operations in digital picture processing. J. ACM 13, 4 (October), 471-494.

Sebestyen, G. [1962]. Decision-Making Processes in Pattern Recognition. The Macmillan Company, New York.

Selfridge, O. G. [1955]. Pattern recognition and modern computers. Proceedings of the 1955 Western Joint Computer Conference, Institute of Radio Engineers, 91-93.

*Shaw, A. C. [1966a]. Pattern recognition bibliography. GSG Memo 4, Computation Group, Stanford Linear Accelerator Center, Stanford, California (January).

Shaw, A. C. [1966b]. Lectures notes on a course in systems programming. Report No. CS 52, Computer Science Department, Stanford University (December).

*Shaw, A.C. [1967a]. A proposed language for the formal description of pictures. GSG Memo 28, Computation Group, Stanford Linear Accelerator Center, Stanford, California (February).

*Shaw, A. C. [1967b]. A picture calculus--further definitions and some basic theorems. GSG Memo 46, Computation Group, Stanford Linear Accelerator Center, Stanford, California (June).

Sherman, H. [1959]. A quasi-topological method for machine recognition of line patterns. Proceedings of the International Conference on Information Processing, UNESCO, Paris, 232-238.

Sherman, P. [1966]. FLOWTRACE, a computer program for flowcharting programs. Comm. ACM 9, 12 (December), 845-854.

Shutt, R. (Ed.) [1967]. Bubble and Spark Chambers. Volume II, Academic Press, New York.

Sussenguth, E. H. [1964]. Structure matching in information processing. Ph.D. thesis, Harvard.

Sutherland, W. [1966]. On-line graphical specification of computer procedures. Technical Report 405, Lincoln Laboratory, Massachusetts Institute of Technology (May).

Tippet, J., Berkowitz, D., Clapp, L., Koester, C., and Vanderburgh, Jr., A. (Ed.) [1965]. Optical and Electro-Optical Information Processing. M.I.T. Press, Cambridge, Massachusetts.

Uhr, L. and Vossler, C. [1963]. A pattern-recognition program that generates, evaluates, and adjusts its own operators. In Feigenbaum and Feldman [1963], 251-268.

Unger, S. H. [1959]. Pattern detection and recognition. Proceedings of the IRE 46, 10, 1744-1750.

Van der Lans, J. [1967]. Hummingbird, automatic film digitizers at the Stanford Linear Accelerator Center. Proceedings of the 1967 International Conference on Programming for Flying Spot Devices, Powell, B. W. and Seyboth, P. (Ed.), Max Planck Institut für physik und Astrophysik, Munich (January), 51-61. Also published as SLAC-PUB-251, Stanford Linear Accelerator Center, Stanford, California.

Warshall, S. [1961]. A syntax directed generator. Proceedings of the AFIPS Eastern Joint Computer Conference, The Macmillan Company, New York, 295-305.

Zahn, C. [1966]. Two-dimensional pattern description and recognition via curvaturepoints. SLAC Report No. 70, Stanford Linear Accelerator Center, Stanford, California (December).