# Alpha-Numeric Character Recognition Using Local Operations

J. S. BOMBA†

## 1.0 Introduction

THIS PAPER describes a demonstration of the recognition of thirty-four alpha-numeric characters. The IBM 704 EDPM was used as a tool to study the method which led to this demonstration. The Generalized Scanner[1] was used as an input transducer for this study.

The method of character recognition which was used here is to extract from the character its "essential" features and then recognize it from these features. For this study such features as horizontal, vertical, and slant straight lines, and intersections of lines have been used.

These features have been extracted by means of local operations. A local operation is a transformation which produces a small section of a new pattern, or field, from data in a corresponding small section of the original pattern (Fig. 1). An entire pattern is transformed by considering *all* of the small sections, where each section is considered independently. Here, the pattern area was divided into a sixty by ninety array of bits (*i.e.* either black or white spots) in order to quantize the visual impression of a character. A typical local operation section would consist of fifteen bits.

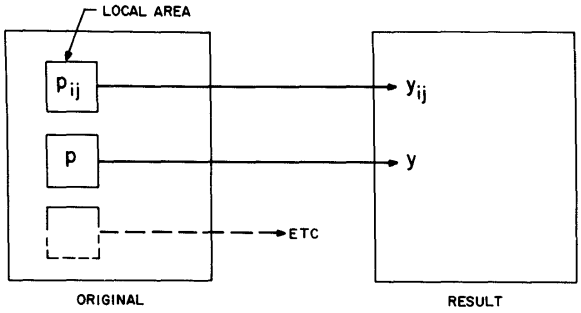


Fig. 1—Illustrated definition of a local operation. A "local area" is a configuration of spots not necessarily square as drawn which is examined by the program. The position $p_{ij}$ of the local area is defined as the position of one particular spot in the configuration; e.g., point $p$ in figure 8 and $y_{ij}$ is the spot which is made into a "1" if the spots found in the local area indicate the presence of a specific feature. Both $p_{ij}$ and $y_{ij}$ have the same coordinates $(i, j)$. Since $p_{ij}$ scans all points in the original patterns, the resulting pattern must contain the same number of points as does the original.

In order to extract a specified feature, the whole pattern is processed and the resulting pattern then is blank unless the desired feature was present in the original pattern.

† Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey.

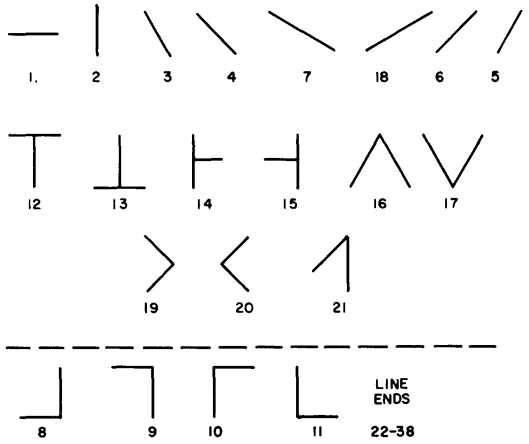


Fig. 2—Features extracted by Feastract. Only features above the dashed line were used for recognition. The numbers are to identify the features for Fig. 12.

The procedure for recognition is as follows. *First*, the characters are processed by a program which reduces the noise in the field by the method of local averaging[2]. In this method, the value of the majority of spots in a three by three rectangular local area dictates the value of the center spot in the new pattern. This process is more effective if it is repeated twice. *Second*, the line width is standardized. Since the matrix granularity is such that a typical line trace as written is always greater than four matrix elements thick, a line-width standardizing operation can extract the middle four matrix locations. *Third*, the features as shown in Fig. 2 are extracted. These are:

(a) straight lines which are horizontal and vertical, and slant straight lines which are at ± 45 degrees, ± 30 degrees, and ± 60 degrees from vertical:

(b) all four orientations of $T$- and $L$-intersections, and

(c) selected orientations of $V$-intersections.

The local areas (as defined in Fig. 1) which are used to extract these features, have the same shape as the desired feature. There are seventeen different features for which the original pattern is examined. In effect, a complete pass is made through the pattern for each feature and whenever a local area in the pattern matches the interrogation local area, a mark is made in a corresponding output matrix. This is done for each of the seventeen different features, thus generat-

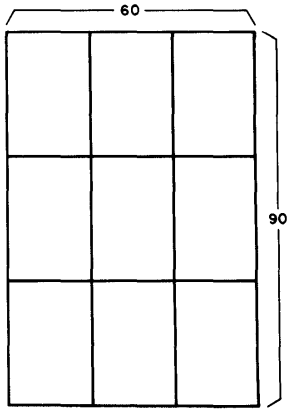


Fig. 3—Division of pattern field. The field consists of a 60 x 90 array of bits.

ing seventeen "new pattern fields." *Fourth,* these fields are divided into nine equal rectangular areas (shown in Fig. 3). The prevalence of spots in certain areas is used to indicate the general position of the feature (*i.e.* top or bottom) and its significance (*i.e.* whether a straight line is long or short). *Fifth,* the recognition is done from the detection of the presence of features. In most cases, it was only necessary to ask whether a given feature was extracted. However, with the limited variety of feature types which were used in evaluating the method, it was necessary to ask further questions about a few features such as horizontal and vertical straight lines. The identification was done with combinational logic. The logic was extended to allow some variation in character styles.

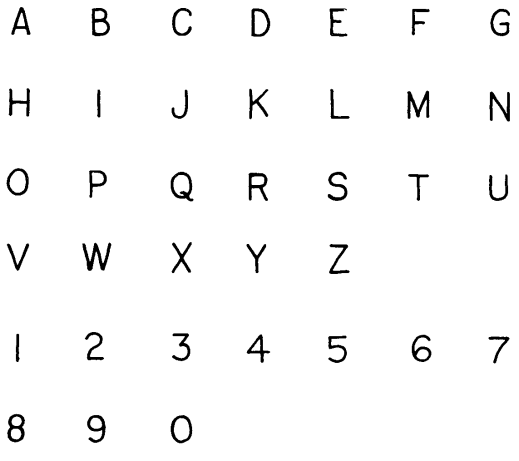


Fig. 4—Typical style of hand printed block capital alphanumerics.

Hand printed, block capital letters and numbers of the style shown in Fig. 4 can be recognized by this method. These are the letters "A" through "Z" and the numberics "1" through "0"; a "one" and "I", and "zero" and "0", are not separately distinguished. The characters may vary from full to one-half size vertically and horizontally. The line widths may vary horizontally and vertically, from a minimum average of four matrix elements to a maximum average of ten matrix elements. The characters must roughly be centered when they are less than two-thirds full size, and must be reasonably free from tilt.

## 2.0 Input Methods

The visual impressions have been transferred into the 704 by a manual and a machine method, both of which are described below (see 2.2 and 2.3).

The visual image is first quantized by dividing the pattern field into square elements and giving each element a value of "0" or "1" depending on whether the field is light or dark at that point. In this study the character lines were dark on a light background, and these character lines are represented by the bit locations which are ones.

The image, which then consists of "0"s and "1"s, is stored in the computer in this binary matrix form.

Each 60 element row of the matrix is split in half and each half is stored in the least 30 significant bits of a 704 core word. The rows are in 90 consecutive pairs of storage locations.

### *2.01 Matrix Size*

A 60 × 90, 5400-element matrix was chosen because:

(1) It is large enough so that quantizing errors should be negligible.

(2) It is large enough so that the edges of the pattern need not be covered by the local operation if the character is more or less centered (thus, it was not necessary to program the special cases which occur when the local area is only partially present as would happen at the field edge. As you will see later in the discussion of feature extraction (see Fig. 10), since the largest local area used extends 7 spots from the local area center, the effective size of the matrix is thus 46 × 76.)

(3) Its width will just fit the printer which we use with the 704, (119 available type wheels are used to print the pattern with every other one printing a blank. The blanks are printed to limit the distortion of the field which the printer introduces because (a) in the original quantization the matrix elements are square whereas (b) on the printer the elements are rectangles whose height to width ratio is 6 to 5 when the blanks are printed and 6 to 10 when they are not).

### *2.2 Manual Procedure*

In order to test the programs and to try sample characters before the Generalized Scanner[1] was finished, test patterns and characters were made up on IBM cards.

For test characters, first the true size characters were drawn on translucent paper. They were then made into viewgraphs which were enlarged by a slide projector. The enlarged image was focused on a 60 × 90 rectilinear grid 1.50 × 2.25 feet. Thus, each ¼″ × ¼″ element could be marked if more than half its area were black. IBM cards were then punched from this grid. One card per row was used with "ones" punched to correspond to black elements and nothing punched for white elements. Special program test patterns were made by marking the large grid.

### *2.3 Machine Methods*

With the advent of the Generalized Scanner it became possible to write the characters on opaque paper and transcribe the scan results onto magnetic tape which can be fed to the 704. This machine method makes it possible to quickly and easily process a large number of characters.

## 3.0 Character Preparation by Local Operations

### *3.1 Noise Reduction*

In order to allow as inputs to the feature extraction program, characters which have missing spots within the line of the character, extra spots in the field outside the line of the character, and fluctuations along the edges of the character lines, the character can be processed by a program which performs local averaging[2]. This program uses a local area, Fig. 5, which is 3 × 3 rectangle. The criteria which are applied for this local area are: (1) if there are five or more "1"s in the nine possible spots and if the center spot was a zero, then the center spot is changed to a one. (2) If there are five or more "zeroes" and if the center spot was a one then the center spot is changed to a zero. (3) In all other cases, the center spot is retained as it was in the original pattern. This process works most satisfactorily if it is repeated two or three times. That is, the original pattern is processed, then the result of the first processing is used as an input to the second process, etc. An example of the results of reducing the noise by this method can be seen by comparing a typical original pattern, Fig. 6, and the resulting pattern (processed twice), Fig. 7.

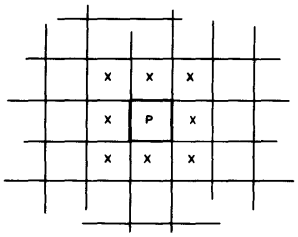


Fig. 5—Local area for local averaging (3 x 3 array).

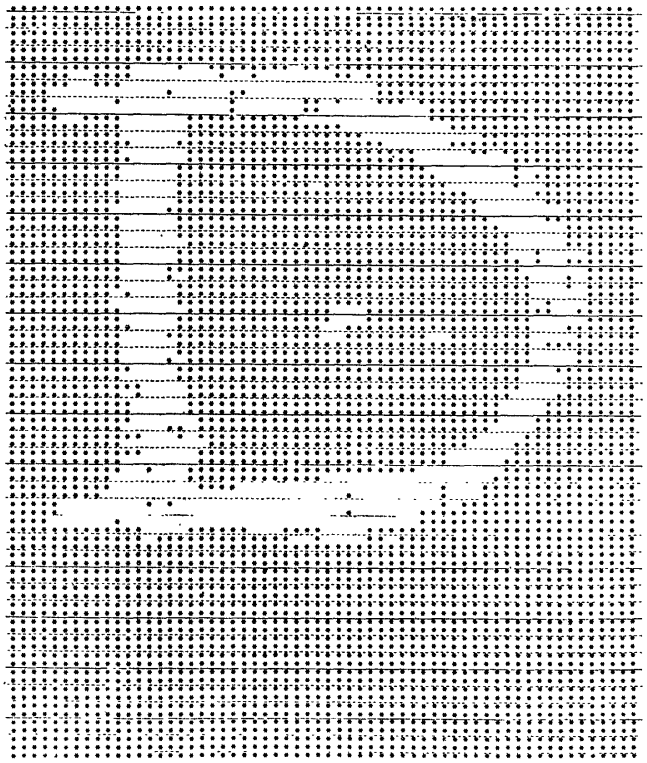

Fig. 6—Unprocessed character "D''.

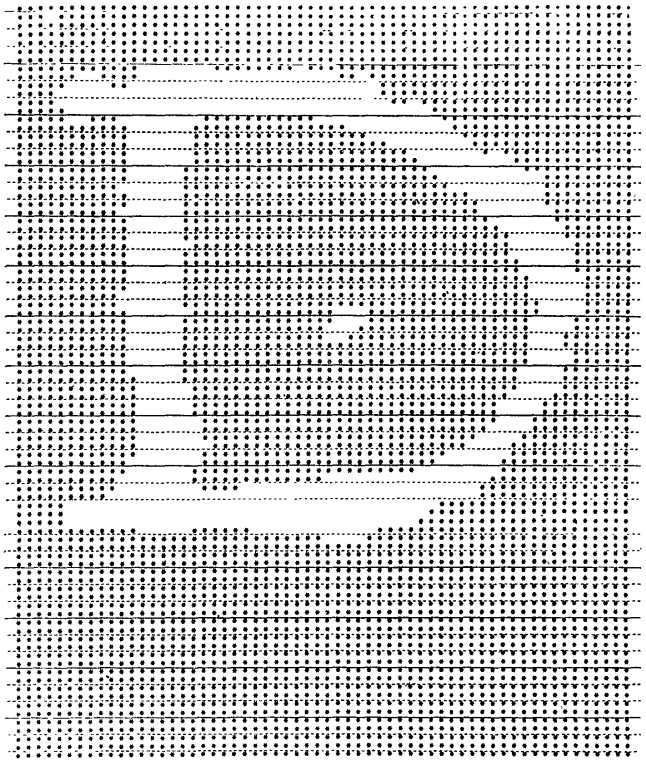

Fig. 7—Character "D" after local averaging twice.

Other criteria for the 3 × 3 local area and a 5 × 5 local area with a variety of criteria were tested. The 3 × 3 local area and the criteria of the previous paragraph were found to be the most satisfactory of the possibilities which were studied.

### *3.2 Line Width Standardization*

The feature extraction program works best when the characters have a uniform line width. Therefore, a program was written to process characters with line widths which vary from 4 to 10 spots and produce characters with standard line widths. This program will change the character lines to a uniform average width of approximately 4 spots. The local area which is used by this program is shown in Fig. 8.

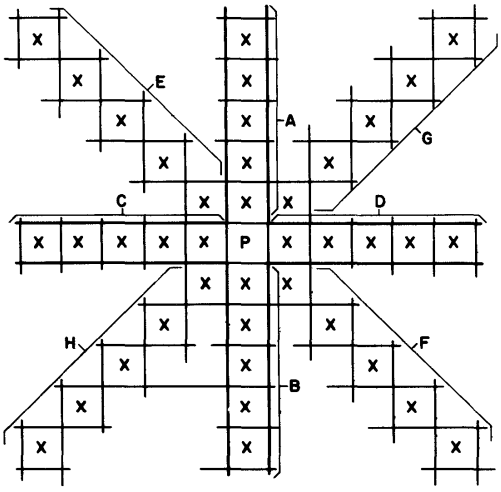


Fig. 8—Local area for line width standardization-thinning (11 x 11 array).

The criteria for this operation are that (1) a spot will be a one only if the original spot were a one and the difference between areas A and B and between areas C and D in the local area was less than 4, or (2) the difference between the number of spots in the areas E and F and the areas G and H are less than 4. An example of "thinning" the previously "denoised" characters is shown in Fig. 9.

This program might be improved if we made the actual radial distances in the local area all equal. (It is obvious, Fig. 8, that five elements at 45 degrees extend further from the center than do five elements in a rectilinear direction).

## 4.0 Feature Extraction

The crux of this method of character recognition are the local operations which are performed by a program which is called Feastract. This program extracts the features which are shown in Fig. 2. Because this program was designed to study the method, it actually extracts more features than are used by the recognition procedure. The most useful features are the long straight lines: horizontal, vertical, and slanted.

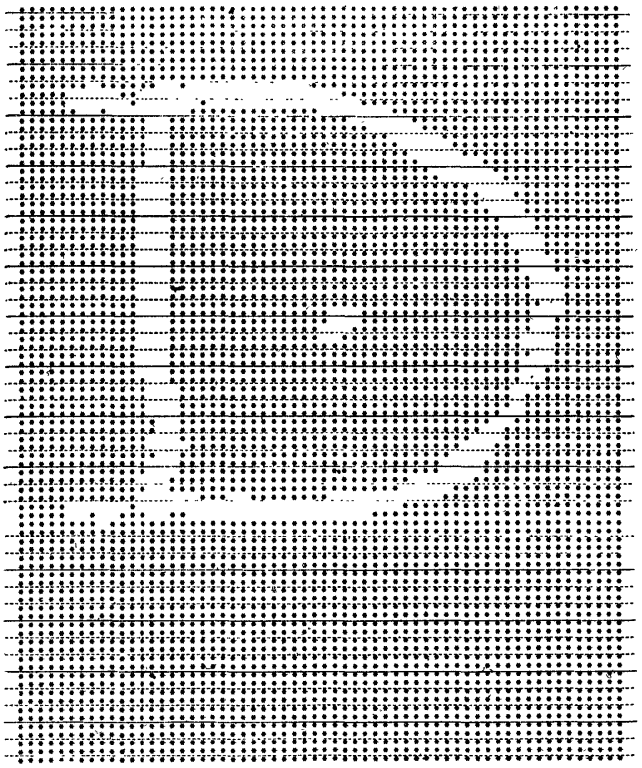

Fig. 9—Character "D" after local averaging twice *and* thinning.

Fig. 2 indicates which of the features are used and which are not.

### *4.1 Local Area*

The local area which is used for the feature extraction is shown in Fig. 10. This local area is actually a combination of a number of local areas, each of which might be used for a different local operation to

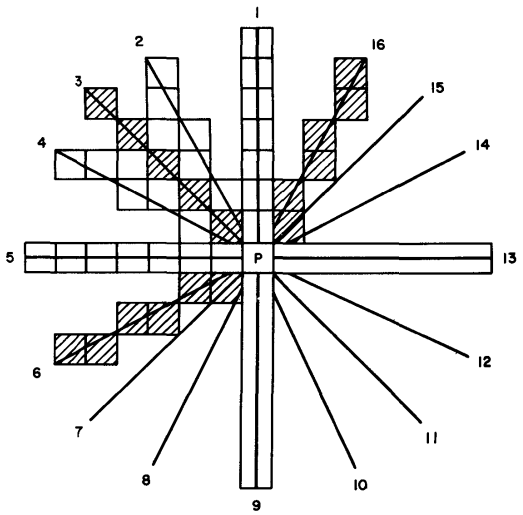


Fig. 10—Local area for Feastract. (1) Shaded areas are to distinquish radii. (2) Matrix elements are only drawn in completely for one quadrant. (3) The local area for a single feature would consist of selected radii. Thus, for a horizontal line, radii 5 and 13 plus *p* would be the local area.

extract a different feature. However, for purposes of study, it was convenient to arrange a local area in the manner shown, a radial pattern which enabled me to vary both the criterion for extraction and the shapes of the features which are found to be most suitable.

It should be noted here that this local area, as applied in this program, is used for local operations; however, if the number of points at which the local area is used is limited to fewer than all points in the overall pattern, then this local area falls into the class of feature extraction methods which can be called radial or polar scan methods[3]. There is one significant difference, however, between the polar scans and this scan. That is, each radius in my local area is required to contain "a" spots (where "a" = the number of spots in the entire radius, or nearly all spots to allow for some noise) whereas in the polar scans each radius is only considered from the viewpoint of crossing a line. That is if any radius contains any spot, it is considered to have crossed the line for the polar scans.

*4.2 Effective Operation of Feastract Program*

In order to clearly present the feature extraction process, the details of the Feastract program will be omitted. Thus, we will discuss the feature extraction process.

Let us consider a specific feature as an example, namely an L-intersection. The local area for an L-intersection consists of radii 1 and 13 and the spot "p" as shown in Fig. 10. The spot "p" with its associated spots is moved over the pattern field in a scanning manner. Whenever all spots in this local area are found to be "ones", a one is written in a buffer image for the L-feature at the same coordinates which "p" has. Since the original pattern is not changed during the above process, the entire operation could, of course, be done in parallel.

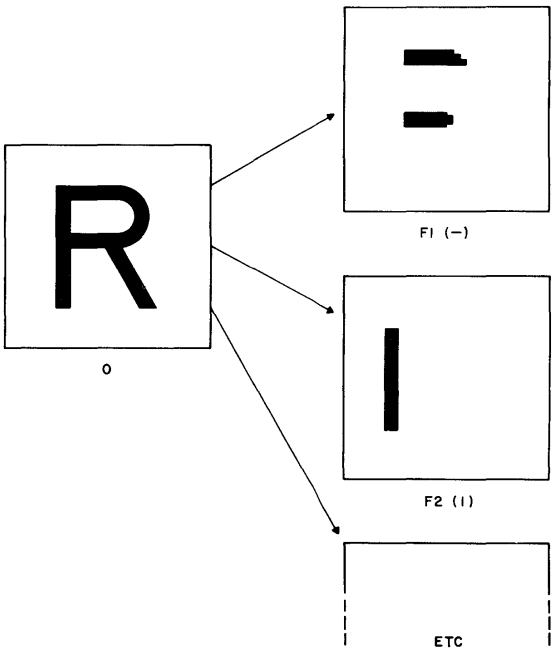


Fig. 11—Original pattern (0) and buffer images (F1, F2, etc.) which result from feature extraction process.

The same process can now be done for each feature which we wish to extract.

The result is a group of buffer images, one for each feature. Fig. 11 illustrates schematically some buffer images which would be produced from the original pattern. Each buffer image contains one feature.

After all of the spots in the original pattern have been examined and the results of the feature extraction process stored in various pattern buffer images, the program proceeds to examine each of these buffer images, and it determines whether or not there were any spots corresponding to the given feature in each image. If a spot is found in a buffer image, the program determines in which of nine areas the spot lies, as shown in Fig. 3. A series of counters are used to remember how many spots lie in each area of the field for the feature under question at the time. When the buffer image for a given feature has been completely scanned and the number of spots in each area of the field have been recorded in the counters, then the result is transferred to the recognition program and also printed out. The basic program was designed so that it would handle up to 50 features.

Note that this feature extraction program does not consider the feature "closed loops" (as encountered in a well-formed B). Although closed loops can be extracted with local operations[4], they are not a necessary feature for recognition of the symbols which were used here.

*4.3 Output from Feastract*

Typical information which is transferred from Feastract to the recognition program is shown in the following list for one feature. This list consists of — the identifying feature number and — the number of spots for the given feature which were found in each area of the field (Fig. 3).

| | *Register Contents* |
|---|---|
| Feature Number | 2 |
| Top Right | 27 |
| Middle Right | 78 |
| Bottom Right | 24 |
| Top Middle | 0 |
| Middle Middle | 0 |
| Bottom Middle | 0 |
| Top Left | 30 |
| Middle Left | 59 |
| Bottom Left | 16 |

These numbers resulted from the extraction of vertical lines (Feature Number = 2) for an H.

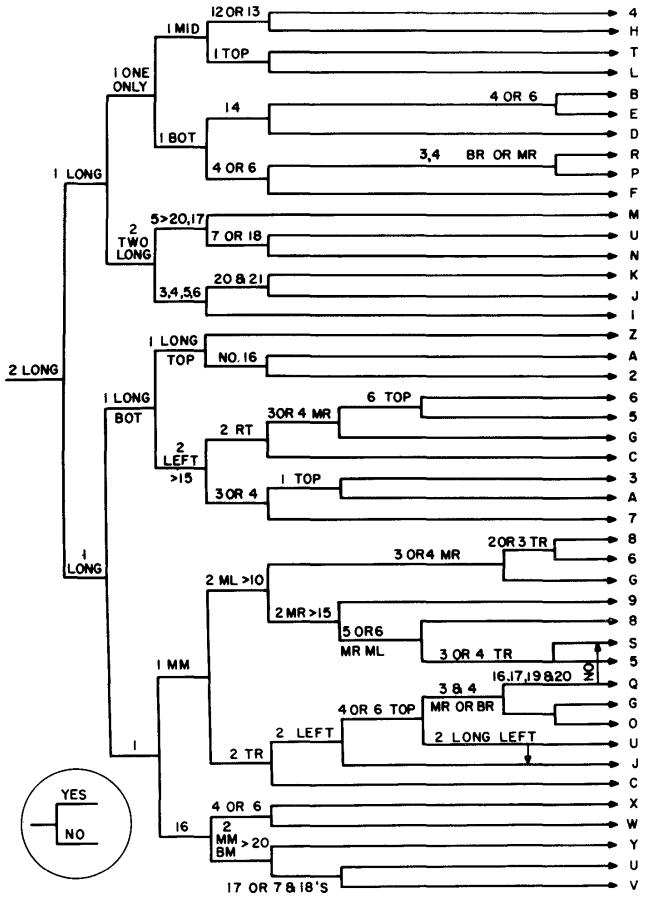


Fig. 12—Recognition tree — logic diagram. The feature numbers at each branch point are identified in Fig. 2. At each branch point the upper branch is followed if the feature is present.

## 5.0 Recognition from Features

Recognition is accomplished by combinational logic with the features as the input variables. Since there is a two-way choice corresponding to the presence or absence of each variable, the logic takes the form of a logical tree (Fig. 12). However, before the actual recognition takes place, certain preprocessing can be done on the features and their locations.

### *5.1 Preprocessing*

A series of subroutines were written to take the information from Feastract and to partially interpret this information to answer such questions as, "how long is a horizontal line"? The detailed specification of said question would be, "does a horizontal line cross three areas in a horizontal row in the field"?

We can now consider more details of the answer to the above question. A general statement of the criteria for answering the question is: "Horizontal lines are considered to be long, if all three areas of the field in a horizontal row each contain more than two marks which were identified from horizontal feature sections." In other words, first, all those areas with only two marks indicated in the new or transformed pattern are eliminated. Second, a test is made to see if there are horizontal features in the three top areas. The same thing is repeated for the three middle areas and then for the three bottom areas. The results are stored in registers called TOP, MIDDL and BOTOM. Third, it is then possible to say that if the TOP register has the number three in it, there is a long horizontal line at the top. When there is a number one in it, then there is a short horizontal segment in the top. Thus, the number and length of horizontal lines can be determined for characters which have the restrictions as indicated in Section 1.0.

The same procedure which was used for horizontal lines can be applied to vertical lines. In this case, the information is stored in registers called LEFT, MIDLE and RIGHT.

There is also a subroutine to determine if there is, for instance, only one horizontal line in the field. The same procedure could be applied to slant lines. However, for this particular recognition tree, it was not found to be necessary to apply this type of preprocessing to other than horizontal and vertical lines.

### *5.2 Recognition*

As indicated in Section 5.0, the recognition proceeds as a series of binary choices. The majority of tests which were used to make these choices are the following two:

(1) The presence of a feature is determined by testing the computer memory location of the feature number for non-zero. (See Section 4.3).

(2) The presence of a feature in one of the nine field areas is likewise determined, except that, in some cases, a minimum number of spots is required in the test. The other tests which were used are discussed in Section 5.1.

When a terminal branch of the recognition tree is reached, the identified character is printed.

It is interesting to note in Fig. 12 that a great many more characters result from all of the tree branches than just the 34 which we initially started with.

We thus have evidence from this tested recognition tree that, even with the restricted input patterns which we have used, redundancy has to be included in the recognition logic. Some subtle variations in characters have only become apparent when they were encountered in testing this recognition logic.

## 6.0 Extensions of This Character Recognition Approach

This character recognition approach, which I have described, has been used to actually recognize characters in order to show that it works. Here, in this feasibility demonstration, only a few of the features which can be extracted with local operations have been used. It seems clear that if the local area of Feastract

were used for other features which it can extract (ends, curved segments), many of the restrictions which have been here (Section 1) imposed on the characters could be removed. Along this same line, if other local operations, such as concavities as described by Unger,[4] were included, the field would not have to be divided and thus larger variations in the characters could be admitted.

Unfortunately, there does not seem to be any good orderly way to specify the most desirable features or to specify the local area or local operation to extract a given feature. Fortunately we can use the 704 to test possibilities without the long process of building hardware. Nevertheless this is still a trial and error process.

Also, the logic for the recognition for this demonstration allows only limited variations in the characters. Although to the human being, characters of a wide variety appear to be the same, very often these characters are actually quite different from their feature viewpoint. Seriphs, slanted characters, and decorations tend to make the character into a different symbol. Thus, in an identification procedure which utilizes features, whether they be the normal features as specified by a human being or some special parameters which are not normally recognized by human beings, these various forms must be treated as separate symbols. The identification procedure could be lengthened to allow other forms of characters to be recognized.

Finally, unless self-organizing systems rapidly become a lot more sophisticated than they are now, any machine which we build to recognize large alphabets of characters of unspecified style will probably use a combination of several recognition methods, along with feedback to make more and more complicated tests until there is a high enough probability that there is a correct identification or a rejection.

## 7.0 Conclusions

It has been explained how features may be detected to recognize certain isolated alpha-numeric characters by using the local operations mentioned above.

Three sample alphabets were tested on an IBM 704, and the characters in these alphabets were separately recognized. With this limited amount of test data, it was felt that any attempt to specify allowable signal-to-noise ratio, character distortion (such as tilt) or a recognition error-rate would not be meaningful.

Others[4,5] have shown the power of using other local operations for this same purpose. Only a few of the many possible local operations have been investigated, and it seems likely that this approach will be useful not only for character detection but also in other areas of pattern recognition.

## 8.0 References

[1] W. H. Highleyman and L. A. Kamentsky "A Generalized Scanner for Pattern and Character Recognition Studies", PROC. W. J. C. C., pp. 291–294, March 1959.

[2] G. P. Dineen "Programming Pattern Recognition", PROC. W. J. C. C., pp. 94–100, 1955.

[3] T. L. Dimond "Devices for Reading Handwritten Characters", PROC. E. J. C. C., pp. 232–237, 1957.

[4] S. H. Unger "Pattern Detection and Recognition", PROC. I. R. E., vol. 47, pp. 1737–1752, October 1959.

[5] R. L. Grimsdale, et al, "A System for the Automatic Recognition of Patterns", PROC. I. E. E., vol. 106, Part B., No. 26, pp. 210–221, March 1959.