Mathematical Equation Recognition : Converting Bitmap To Text

By

Dipankar Saha (MTC9611)

Under the supervision of

Dr. B. Chanda
Electronics and Communication Science Unit.
INDIAN STATISTICAL INSTITUTE
203 , Barrackpore Trunk Road
Calcutta - 700035

1998

Certificate of Approval

This is to certify that the dissertation work entitled **Mathematical Equation Recognition : Converting Bitmap To Text** submitted by Dipankar Saha , in partial fulfillment of the requirements for M.Tech in *Computer Science* degree of the *Indian Statistical Institute* , is an acceptable work for the award of the degree.

Date :

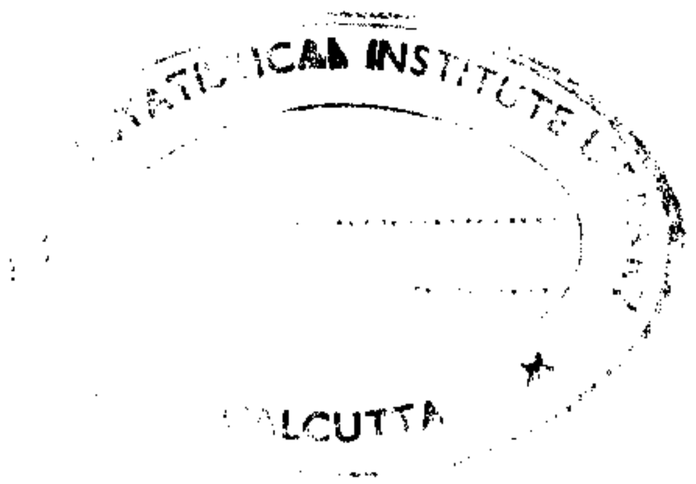
29th July 1998

B. Chaudhuri

(Supervisor)

P. K Nandi 29/7/98

(External Examiner)



Acknowledgement

I owe a debt of gratitude to Prof. Bhabotosh Chanda (*Dept. of E.C.S.U.,I.S.I.*) for providing me invaluable technical support throughout the project . During the course of this project , he has always given me helpful suggestions whenever I was in trouble and finally helped me to prepare this manuscript.I would also like to acknowledge Mr. Subhomoy Mitra (*Dept. of C.S.S.C. , I.S.I.*) for giving some innovative ideas to tackle the problem.

Finally I would like to thank all my teachers and batchmates for their support throughout the project.

Abstract

Transforming a paper document to its electronic version in a form suitable for efficient storage,retrieval and interpretation continues to be a challenging problem. After pixel level processing and segmentation technique has segmented the image of a paper into three basic components namely Text, Image or Drawing and Mathematical expression. A lot of post processing is required to store the datas in some particular format so that it can easily be retrieved and at the same time huge lossless compression ratio is achieved. In this project we are dealing with the post processing of Mathematical expression. Our basic objective here is to generate the Latex code automatically,which will give us back to that Mathematical expression. For that, we first used ***Connected Component*** techniques to seperate out the characters and ***Template Matching*** techniques to recognize the mathematical symbols and finally proposed a novel application of a ***m-Ary Tree*** to generate the Latex code. Here *m-Ary tree* model has been constructed based upon the *relative positions* of the symbols.

Key Words : *Connected Component, Component Levelling, Template Matching, m-Ary Tree, Preorder Traversal.*

Contents

1 Introduction

Over the years transformation of a paper document to its electronic versions and subsequent storing into some efficient format have become an important and challenging application domain for computer vision and image processing researchers. In writting research paper **Latex** is used as an universally accepted tool. Hence we can safely assume that the published research papers are always written in **Latex** and they always conform to the **Latex Syntax.** Storing a image in the form of a scanned binary image takes a huge amount of space. Instead if we can process that binary image and automatically generate the **Latex** code which was actually used for preparing the concerned paper, then our problem is solved. But this seems to be an oversimplified version of the original problem. Considering the Latex Syntactic structure,to have any amount of success regarding this matter,first thing that we have to do is to segment the Binary Image into three distinct regions i) **Text ii) Drawing/Image iii) Mathematical** region. This itself is a very complicated job, considering the huge number of approaches that are available. Hence for the sake of simplicity we ignore that segmentation part completely and start our job by assuming that a perfectly segmented mathematical region is given as an input to us.Our work concentrates on recognising the different distinct components(mathematical symbols) in that region and finding out the syntactic binding among them based upon their relative positions.Due to problem of perfect segmentation we are again narrowing our path by taking a **Synthetic Image** as our input. This image is generated by compiling a **Tex file**, using **dvips** application on the **dvi** file generated in the previous step, and finally applying **gs** application on the **postscript** file

generated in the previous step. It gives us two fold advantages. Firstly this image is devoid of any amount of **noise** or **skewness** which otherwise we had to remove by preprocessing that given image. Secondly it gives us the **Ideal** character which paves the way of easier **Template Matching** for their recognition. But this fundamental assumption does not reduce the gravity of our work in any great extent, as, otherwise only **optical character recognition** part would have been hampered to an extent. From the point of view of practical application of this project to any significant amount, it is very difficult to design a **fully automated system** for document transformation and that too with a very **high percentage of accuracy.** This is mainly due to the fact that most of the algorithms developed for the second phase i.e. code generation part,depends on the **distance heuristics** which might vary **significantly** from document to document. Hence rigorous training is always necessary to fix up the heuristics to find the best possible result. But under any circumstances it can never give a hundred percent gurantee of getting back the original code. A **Data Flow Diagram** of the whole process is given in **Fig. 1**. Deatils of these processes are described later on.

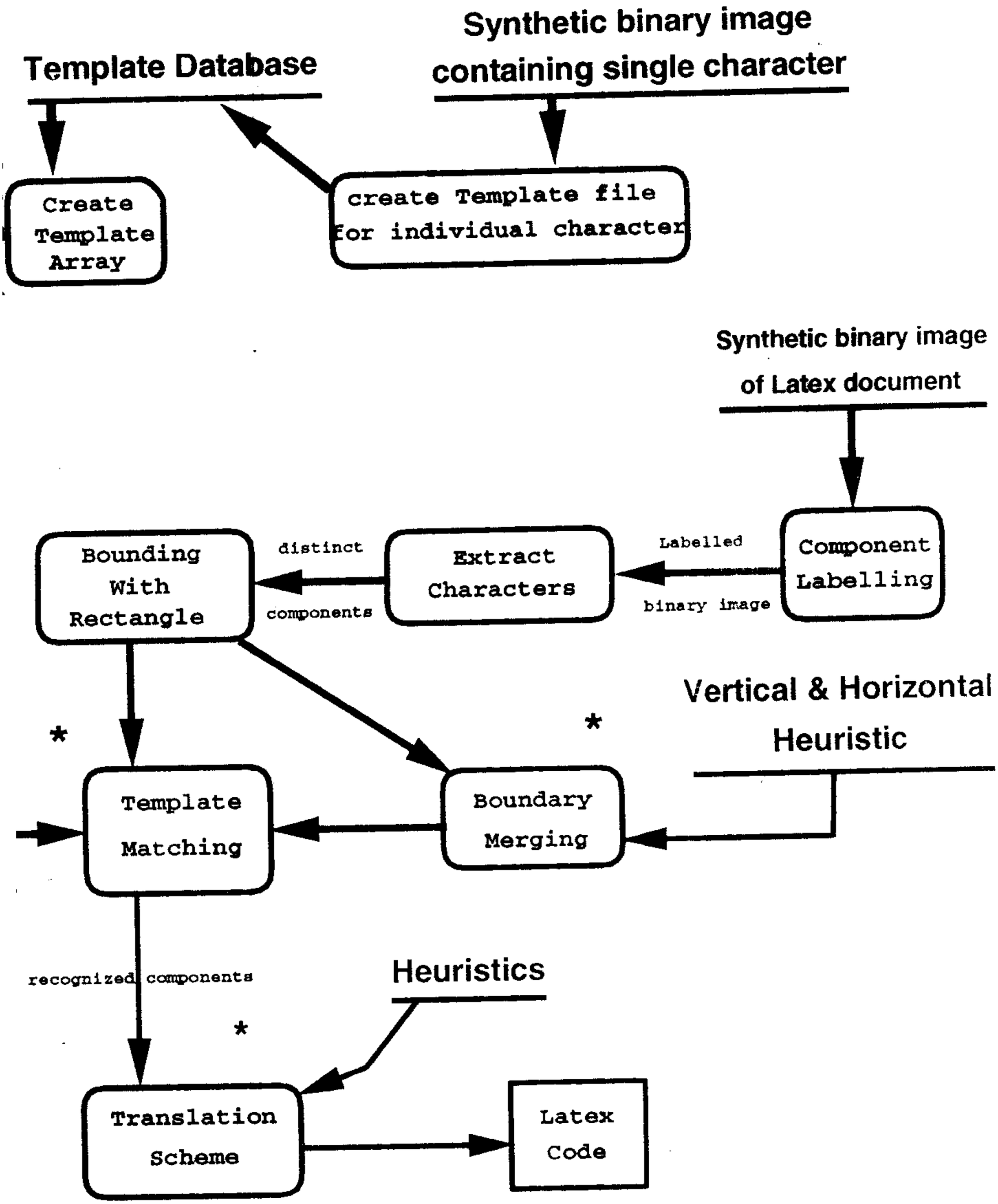


Fig. 1. Data Flow Diagram

2 Problem Description

As stated earlier, the goal is to generate the Latex code from the bitmap image of thr region. As an example, consider the following equation in Fig. 2a.

$$\sum_{i=0}^{\infty} x_i = \prod_{i=0}^{q} y^i - \beta$$

Fig 2a.

The **Latex** code corresponding to the above equation is :

```
$$
\sum_{i=0}^{\infty}x_{i}=\prod{i=0}^{q}y^{i}-\beta
$$
```

In this respect,now we provide a brief description of the different modes of Latex.Here there are altogether three modes. (i) Text (ii) Image/Drawing and (iii) Math. Our subject of interest is only the Math mode.

A formula that appears in the running text, called an intext formula. is produced by **Math** environment. It can be invoked with either of the two short forms, \(...\) or $$ ... $$ and with long forms *\begin \end* combination. A numbered displayed formula is produced by the equation array put into *Math* mode. In this mode it ignores spaces in the input. For giving spaces in between we have to put & or ~ before the symbol concerned. Now we provide examples of some common structures.

Subscripts & Superscripts :

x^{2y} : x^{2y}

x_{2y} : x_{2y}

x^{y^2} : x^{y^{2}}

x^{y_1} : x^{y_{1}}

Fraction :

$x = \frac{y+\frac{z}{2}}{y^2+1}$: x = \frac{y+\frac{z}{2}}{y^{2}+1}

$\frac{x+y}{1+\frac{y}{z+1}}$: \frac{x+y}{1+\frac{y}{z+1}}

Roots :

$\sqrt{x+y}$: \sqrt{x+y}

$\sqrt[n]{2}$: \sqrt [n]{2}

Ellipsis:

$x_1,\ldots,x_n$: x_{1},\ldots , x_{n}

$a_1 + \ldots + a_n$: a_{1} + \ddots + a_{n}

Similarly differnt types of symbols are there.

Normal alphabets : Lowercase , Uppercase eg. a,A,xy,Y etc

Greek letters : Lowercase and Uppercase eg. α, ω, Ω etc.

Binary operation symbols : eg. $\pm, \times, \div$ etc.

Relational symbols : eg. $\leq, \equiv, \propto$ etc.

Arrow symbols : eg. $\leftarrow, \Leftrightarrow$ etc.

Misc. symbols : eg. ∇, ∂, ∞ etc.

Variable size simbols : $\sum, \int, \oint$ etc.

All the above symbols can be written in different styles like bold, italic, roman, sans serief type writter. But in our problem we have taken into account of only a single style symbols. This is also our one of the assumptions. But it can be easily extended by creating data base for all possible styles.

Considering the above facts, to achieve our goal, first job is to seperate out the different characters present in the bitmap. Secondly, we have to recognize each character and find their relative positions. In order to do that we have to find the parameters needed to describe the boundary rectangles of each extracted symbols. Finally we have to device an effective translation scheme which will operate based upon their relative positions and will give back the **Latex** code.

Assumptions :

1. Input image is a **synthetic** image. It differs from the real life scanned image of the document in a sense that any sort of noise and **skewness** is totllly absent from it.

2. Different distinct **8-Connected** components i.e. distinct characters should not be merged in the bitmap image. If they remain so then our algorithm will fail to seperate them out and consequently recognition system will fail.

3. Resolution of **dvips** utilityshould always be set at **600 dpi**. Our standard template is prepared only on that resolution. For other resolution they will differ, resulting the failing of the recognition scheme.

4. Our **Template database** is limited at present. It does not contain all possible **characters, types** and **fonts**. For future extension they can be incorporated.

3 Proposed Method

Our proposed solution to the afforementioned problem consisits the following steps :

1. Generation of standard Template database.Character recognition.

2. Seperate out each character.

3. Determining bounding rectangle of each character and parameters required to determine their relative positions.

4. Merging of bounding rectangles.

5. Character recognition.

6. Translation scheme.

3.1 Generation of standard Template database :

Steps are following :

a> Create a bitmap file which consists of a single character whose *Template* is to be stored.

b> Find a bounding rectangle of that charcater. Get the parameters needed to describe the rectangle i.e. width and height. **c>** Extract the binary bitmap of that rectangle and store in a file along with its height and width.

d> Name the file same as the name of the character. Its extension is .chr.For eg. let us take a look at the file ***a.chr*** which is the Template file of 'a'. It is shown in a table structure.

6	6				
0	0	0	1	0	0
0	1	1	0	1	0
1	1	0	0	1	0
1	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	1	1

Here 1st row contais number of rows and number of columns. From next row it contains the bitmap of the character 'a' taken at **600 dpi** resolution.

e> Create all the files of the characters in this way.

f> During the execution of the program store this database in the active memory to reduce comparison time.For that, create an array of structures whose size is equal to the number of characters in the database.

g> Each structure contains name,height, width and bitmap as its field.

h> Initialize the database at the beginning of the program by filling up this structures by reading from the corresponding files.

3.2 Seperate out each character :

For separation of characters we need the concepts of *Image Segmentation*. Image segmentation refers to the decomposition of a scene into its components. It is a key step in image analysis. For example, an application like this would first segment the various characters before preceeding to identify them. Ideally a partition represents an object or part of an object. Formally, segmentation can be defined as a method to partition an image, $F[i, j]$, into subimages, called regions,P_i, ... P_k, such that each image is an object can-

didate.

Definition 3.2.1

A ***region*** is a subset of an image.

Definition 3.2.2

Segmaentation is grouping pixels into regions,such that

1. $\bigcup_{i=1}^{k} P_i =$ Entire image $\{P_i\}$ is an exhaustive partitioning.

2. $P_i \bigcap P_j = \phi, i \neq j, \{P_i\}$is an exhaustive partition.

3. Each region $\{P_i\}$ satisfies a predicate;that is,all points of the partition have some common property.

4. Pixels belonging to adjacent regions,when taken jointly,do not satisfy the predicate. **Definition 3.2.3**

Neighbours

A pixel in a digital image is spatially close to several other pixels. In a digital image represented on a square grid. a pixel has a common boundary with four pixels and shares a corner with four additional pixels. We say that two pixels are *four neighbours* if they share a common boundary. Similarly. two pixels are *8-neighbours* if they share at least one corner. For example. the pixel at location $[i, j]$has 4-neighbours $[i+1, j], [i-1, j], [i, j-1], [i, j+1]$. The 8-neighbours of the pixel include the 4-neighbours plus $[i+1, j+1], [i+1, j-1], [i-1, j+1], [i-1, j-1]$. For detail illustration look at the following fig.

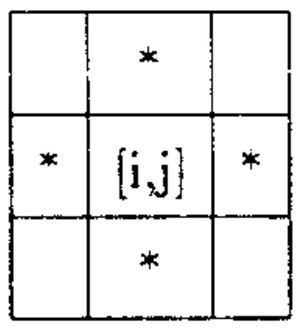


Fig.3.1a(4-neighbours)

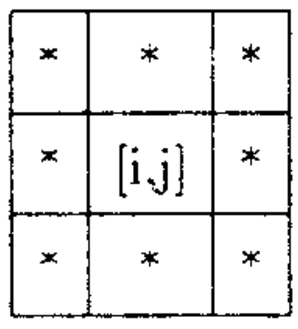


Fig.3.1b(8-neighbours)

Definition 3.2.4

Path

A path from the pixel at $[i_0, j_0]$ to the pixel at $[i_k, j_k]$ is a sequence of pixel indices $[i_0, j_0]$, $[i_1, j_1]$, $[i_2, j_2]$ .....$[i_{k+1}, j_{k+1}]$ such that pixel at $[i_0, j_0]$ for all k with $0 \leq k \leq n-1$. If the neighbour relation uses 4-connection then it's a 4-path ;for 4-connection 8-path.

Definition 3.2.5

Foreground

The set of all 1 pixels in an image is called the foreground and is denoted by S.

Definition 3.2.6

Connectivity

A pixel $p \in S$ is said to be connected to $q \in S$ if there is a path from $p$to q consisting entirely of pixels of S.Note that connectivity is an equivalent relation. For any three pixels p,q.and $r$in S, we have the following properties :

1. Pixel *p*is connected to *p*(*reflexivity*).

2. If *p*is connected to *q*, then *q*is connected to *p*(*commutivity*).

3. If *p* is connected to *q* and *q* is connected to *r* then *p* is connected to *r*(*transitivity*).

Defination 3.2.7

Connected Components

A set of pixels in which each pixel is connected to all other pixels is called a *connected* component. **Component Labelling**

A component labelling algorithm finds all connected components in a binary image. We use this algorithm to find the binary bitmap of the different characters, their bounding rectangle, size and position. There are two algorithm for component labelling : recursive and nonrecursive. We will use

Sequential Connected Components Algorithm using 8-connectivity.

Steps are following.

Phase 1

a> Scan the image left to right , top to bottom.

b> If the pixel is 1,then

(1) If one of its left or upper three pixels of its 8-neighbours has a label.then copy the label.

(2) If all the pixels which have labels are same then copy that label.

(3) If marked pixels have different label then copy the label of the first pixel traversing clockwise direction starting from the left pixel. Enter this label in the equivalent table as equivalent.

(4) Otherwise assign a new label to this pixel.

c> If there are more pixels to consider. then go to step 2.

end of **Phase 1**

Phase 2

a> Process the equivalent table and find the lowest label for each equivalent set in the equivalence table.

b> Scan the picture as described in step <*a*> of Phase 1. Replace each label by the lowest label in its equivalent set. Here use the equivalence table as a lookup table.

end of **Phase 2**

At he end of Phase 2 each connected component is marked with an unique label. As a result extraction of these components becomes easier.

Implementational Details & Data structures used :

For storing the bitmap image a two dimensional integer array is used. Here integer array is taken to facilitate the component labelling algorithm to work where integer values are assigned to different labels.

For storing the equivalent table linear linked list is used which is dynamically updated in the Phase 1 of the previous algorithm.

3.3 Determining bounding rectangle of each character and parameters required to determine their relative positions

Algorithm for extracting components from the labelled bitmap

Begin List Construction

a> Process the equivalent table to find the number of distinct components present in the image.

b> For each component create a linear linked list which are again stored in a linear array.

c> Assign the unique labels of the equivalent table to each member of the array.

d> Start scanning the image from left to right from top to bottom.

e> If the pixel value is not 0 then create a new list having its own label and add the list to the end of the list in the proper position of the array.Also store the coordinates of the point in the each list corresponding to a point.

f> If more pixels left then go to step d.

End of List Construction

Begin array of structure construction

a> Create an array of structure where array length is equal to the total number of components.

b> From the list corresponding to each component find out the minimum of x and that of y coordinates and also maximum of x and y coordinates.

c> Find the bounding rectangle of the symbol. Store its left top coordinates and height and width as parameters.

d> Fill up this bounding rectangle bitmap. Put 0 everywhere except in those points where 1 exists in the list.

e> Do it for all the lists.

End array of structure construction.

There are however few problems related to character bitmaps. Few characters are vertically disconnected. viz.Fig. 3.2

$$i, j, !, =, \Xi \div \equiv etc.$$

Fig. 3.2

Algorithms so far we have described, will give these symbols as broken ones. Hence in the next statge (i.e. recognition stage) algorithms will not

be able to detect those components properly. Hence we have to merge distinct components depending on the vertical distnace heuristics. Similarly bitmap of some of the components are broken in the horizontal direction. To recognize properly, merging along the horizontal direction depending on the horizontal distance heuristics is necessary. The scheme is following :

a> After extraction,apply matching algorithm on each components. If there is a match, put the name in the symbol name.

b> Else put NULL in the symbol name.

c> Choose a distance heuristics value. Apply vertical merge on those symbols which are yet to be detected(i.e. NULL).

d> If required then apply step ***c>*** again. But this time with a higher heuristics value.

e> If required then apply step ***<c>*** again. But this time with a horizontal distance heuristics and also merge it in the horizontal direction.

3.4 Merging :

It is explained below using Fig 3.3a and Fig 3.3b

0	0	1	0
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	0
0	1	1	0
0	1	0	1
0	1	1	0

Fig.3.3a

1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1

Fig.3.3.b

From the above two pictures it is quite evident that character maps of the two symbols are broken.Component labelling detects them as two seperate components. Hence merging is required. Steps are following.

Begin merging

a> Take the component in the array for processing if the detecting flag shows FALSE.

b> Try to find the components which overlap in the x-direction.

c> Distictly find the top and bottom one.

d> If topLeftCorner.y+height-bottomLeft.y <= yDistanceHeuristics then merge two components by :

1> finding the lowest x and y and also the highest x and y cordinates of the two rectangles.

2> finding the height and width of the new bounadary rectangles by taking out the common areas.

3> filling up this rectangle with the bitmap of the two components.

e>Finally update one structure with these new values and mark the others processing flag in a way such that it is never used.

End merging

In case of horizontal merging all the steps are same except in **<b>** it is

x-direction and in <**d**> all y is replaced by x.

3.5 Character recognition

One direct method of character recognition is Template Matching. The components that we have extracted so far are stored in the two dimensional matrix form. Also our standard *Templates* in the database has been kept in the same fashion along with their sizes. Here we will move one components bitmap over all the *Templates* in the database. If the sizes of the element if the database falls within a range then we compare their bitmaps and find number elements matched. From this statistics we calculate the *confidence of matching.* If it is greater than a pre defined value then we can conclude with that amount of confidence level that this *Template* resembles that component and we copy that name to the component name field. Detail algorithmic steps are given below :

a> For each component in the component array do

b> Get a component and do for each element in the database do *1)* If the size of the component i.e. height and width are equal to height and width of the *Template* or lie within a certain range of that then compare their bitmaps.

2) Count the number of matches in step 1.

3) Find the *confidence of matching* by dividing the counter by the area of boundary rectangle of the component.

4) If *confidence of matching* is greater than a predefined value then copy the *Template* name to the component name and break.

5> Else continue.

c> If there is no match, copy "**NULL**" to the component name.

3.6 Translation scheme

So far we have discussed about the methods for recognizing characters and finding out their relative positions.Relative positions will be obtained by the parameters of the bounding rectangles. Now we are going to discuss how to get back the Latex code from these informations. This is possibly the trickiest and the most vulnarable section of the whole project. Not only because there does not exist any sort of existing algorithm for this but also because of the fact that proposed algorithm heavily depend upon the distance heuristics which may vary widely from ducument to document.

Proposed method is illustrated with the Fig. 3.5.1

Consider the following example :

$\sum_{i=0}^{\infty} x_i^2 = \prod_{j=0}^{10} \beta_j + \alpha^2 k - \alpha\gamma$

Fig. 3.5.1

Applying previous algorithms we have been able to extract :

$\sum ; _{i} ; ^{\infty} ; x ; _{i} ; = ; 0 ; ^{2} ; = ; \prod ; _{j} ; = ; 0 ; ^{10} ; \beta ; _{j} ; + ; \alpha ; ^{2} ; k ; a ; \eta$

But we have no idea about their relative positions and how are they syntactically bound.Finally we have to reach the following code :

$$

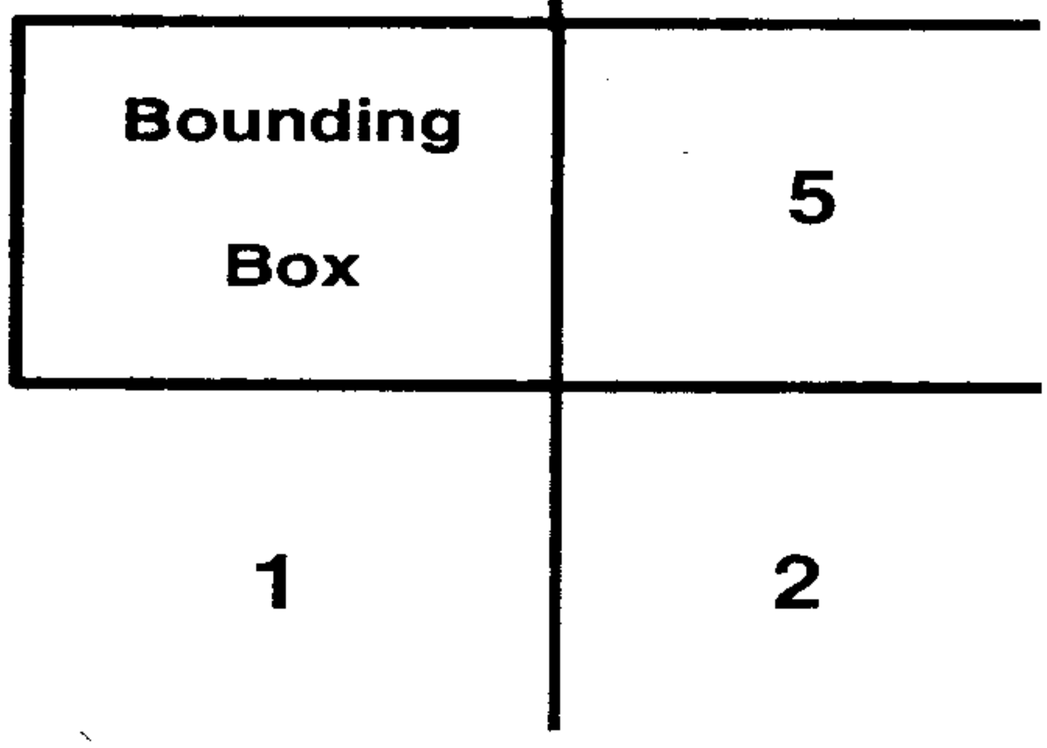
\sum_{i=0}^{\infty}x_{i}^{2}=\prod_{j=0}^{10}\beta_{j}+\alpha^{2}k

- a \eta

$$

This is our objective.

Now we are going to propose a novel application of a ***m-Ary*** tree structure to serve our purpose. We take m = 5. We divide the surroundings of a bounding rectangle into 5 regions as illustrated in the following **Fig. 3.5.2**



It is being observed that if the main symbol lies in the rectangular box(vide Fig. 3.5.2) then subscripts (including boundary rectangle) lie in the Reg. 1 and Reg. 2. Superscripts lie in the Reg. 3 and Reg. 4 and normal symbol lies in the Reg. 5. Here some typical observations are made based upon which certain heuristics have been formed.

Observations :

1> Upper part of the bounding rectangle of Reg.1 symbols always lie at least 1 pixel below the horizontal line above.

2> All the symbols in the Reg. 1 and Reg. 2 are subscripted symbols.

3> All the symbols in the Reg. 3 and Reg. 4 are superscripted symbols.

4> Subscripted symbols starting at the Reg. 2 has their upper boundary in Reg. 5 and lower boundary in Reg 2. Sinilarly right boundary alwats lie to the right of the center rectangle.

5> Lower boundary of the symbols from the Reg. 3 always lie at least 1 pixel above the top boundary of the rectangle.

6> Superscripted symbols starting at the reg. 4 has their upper boundary in Reg. 4 and lower boundary in Reg 5. Sinilarly right boundary alwats lie to the right of the center rectangle.

7> For symbols lying in the Reg. 5 if upper bounadry lies in Reg. 4 then lower boundary eithe touces the extended lower horizontal line or exceeds it. Else if the upper boundary touches the extended upper horizontal line the lower boundary must lie in the Reg. 5 or may touch the extended lower horizontal line .

8> Note that all the above observations made are recursively applicable to each of the bounding rectangle.

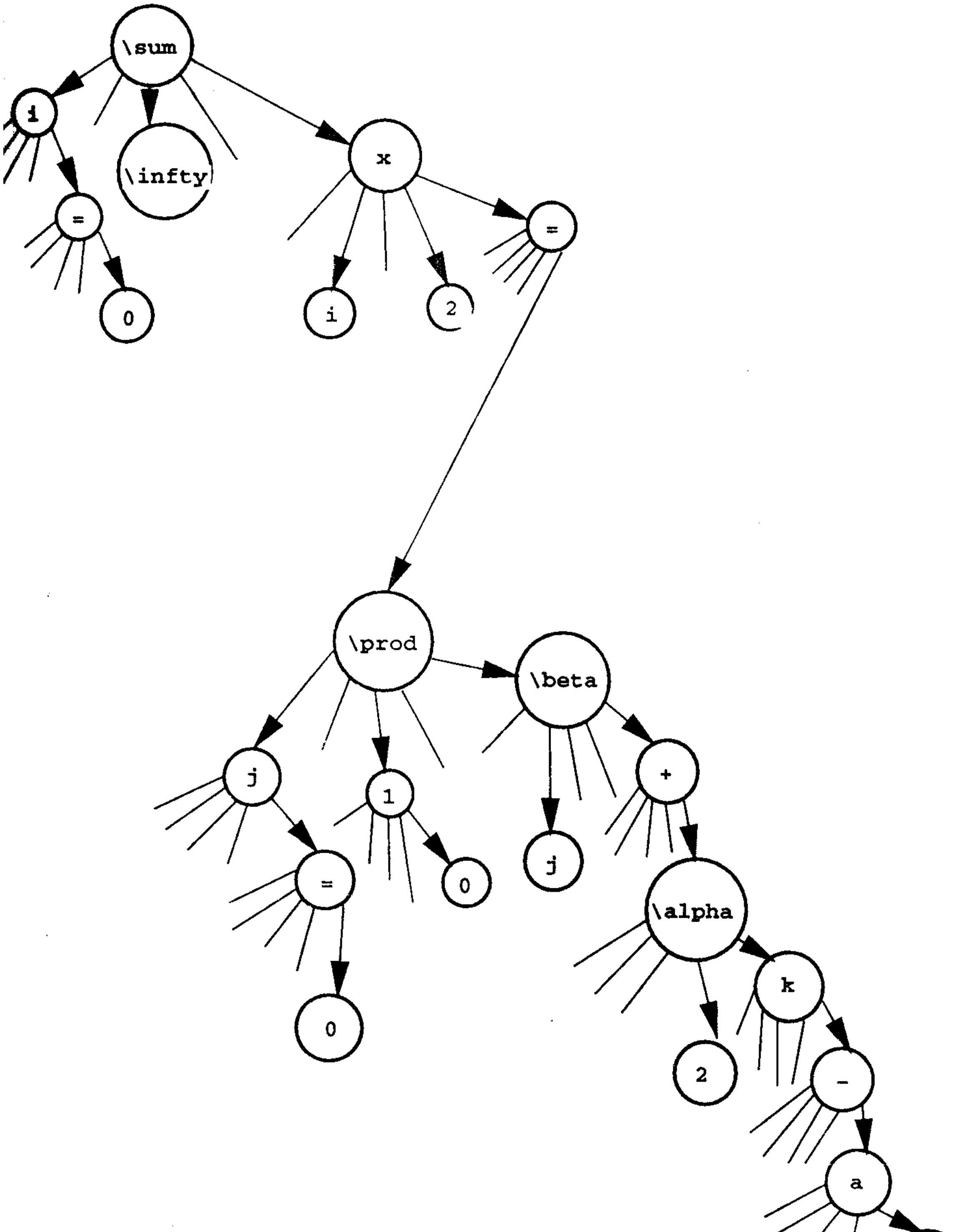
In the proposed ***m-Ary*** tree structure for each reg. a link is stored.If a symbol exists in that reg. then pointer is set to that node else the pointer is made NULL.

Proposed structure of each tree node as follows:

```
Struct _TreeNode{
char cNodeName[10]; /*for storing the name of the symbols*/
int nLinkFlag; /*it is set to -1 at the beginning.It is used for insertion of element in the tree*/
int nTraverseFlag; /*same as above except it is used for preorder like traversal*/
```

```
Point pStartPoint; /* holds the starting co-ordinates of the rectangle*/
int nWidth; /* width of the rectangle*/
    int nHeight; /* height of the rectangle */
    char PrintFlag; /*indicates whether the node has been processed during traversal or not*/
    struct _TreeNode tNext[4]; /*stores the link to the next links of the tree */
    };
```

As said earlier the overall structure is recursive in nature. As a result depth first approach is taken i.e. once a node is reached we try to move as deep as possible by following the link in the increasing order starting from 0. When no more link is left to scan, we reject this node and move one step back to the previous node and process in the same fashion. ***If we are succssesful in creating the tree in this fashion then if we made a preorder traversal of the tree ,barring from '_' ; '^' ; '&' ; '{' ; '}' we will get back the Latex code.*** This is illustrated in the following Fig.



Example of a m–Ary Tree

The above Fig. is for the equation:

$\sum_{i=0}^{\infty} x_i^2 = \prod_{j=0}^{10} \beta_j + \alpha^2 k - a\eta$

If we make a ***preorder traversal*** of the above tree then we will get :

$\sum ; _i ; ^{\infty} ; x; _i; =; 0; ^2; = ; \prod ; _j ; = ; 0 ; ^{10} ; \beta ; _j ; + ; \alpha ; _2 ; k ; a ; \eta$

Actual **Latex** code is :

$$

\sum_{i=0}^{\infty}x_{i}^{2}=\prod_{j=0}^{10}\beta_{j}+\alpha^{2}k

- a \eta

$$

Overall translation scheme is divided into two parts :

a> Insertion into the ***m-Ary*** tree

b> Preorder traversal of the ***m-Ary*** tree

Algorithm for inserting components into the tree :

a> Find out the left most component in the bitmap which is not od suffix type. Do it by comparing the start coordinates of the bounding rectangle and using CFlag which indicates the type of symbol.

b> Create a tree node for this component. Fix it as a root of the ***m-Ary*** tree.

c> Try to find out whether there exists any component in the Reg. 1, 2. 3. and 5 respectively. Searching for component is done in that sequential fashion. If there does'nt exist any component, return the tree. Else go to step <d>.

d> Create a new tree node. Push the previous node into the stack and start processing the new node. Also update the nLink field in each stage. Try to find out the components in the same way described in step <c>.

e> If a component exists, do step d>.

f> Else pop the node(if there exists any) which is present in the stack and start processing for other links. If stack is empty and returns NULL then stop and return the head.

Here components in different directions are found based on the assump-

tions made earlier. Since whole algorithm behaves like depth first search, the processing of the root of the tree occur at the last step.

Algorithm for preorder traversal of the tree :

It is a recursive procedure. Basic principle of inorder traversal is :

1. Process the node r.
2. Do inorder traversal of its link 0
3. Do inorder traversal of its link 1
4. Do inorder traversal of its link 2
5. Do inorder traversal of its link 3
6. Do inorder traversal of its link 4

Steps

a> First take the root node and depending on its flag value print it into the file (eg. if the flag is 'G' put \symbol_name if flag is 'A' put symbol_name). Increment the file pointer accordingly.

b> Do the traversal in the fashion described as above. Use stack to remove inherent recursion of the process.

c> During traversal if there exists a node either in the direction 1 or 2 then put _{ before the symbol name and store the address of the node corresponding to which _{ has been inserted in the file. Push the node into the stack.

d> When all the links of a node is traversed, its processing is totally complete. So pop up node (if there exists any) from the stack. But before popping up, compare whether any { was inserted for the just processed node(as mentioned prviously it is stored somewhere).If true the put } and delete that entry.Else continue.

e> During traversal if there exists a node either in the direction 3 or

4 then put ^{ before the symbol name and store the address of the node corresponding to which ^{ has been inserted in the file. Push the node into the stack.

f> Do same as step <d>.

g> Stop when stack is empty and no more node is there to undergo processing.

4 Experimental Results

4.1 System used/OS

Intel Pentium-133 chip

16Mb RAM

Linux operating system with 70Mb swap space.

4.2 Results Obtained :

It is applied on number of binary images (all of them excludes fraction symbol). Although it can identify the fraction symbol, it can't generate the correct code for that image. As far as character recognition is concerned it is successful in recognizing almost 95% of the characters. But due to problem in the heuristics part, it some times fail to insert the element properly in the tree resulting a code which mismatches with the image and many times disobey the **Latex** syntax. We are now trying to develop an ***adoptive heuristic based technique*** and we expect that this would give a much better result.

5 Discussion and Conclusion

5.1 In our implementation part we have developed the following things :

1. module for extraction of the bitmap for standard templates,

2. module for generating standard database,

3. module for implementing component labelling algorithm.

4. module for extracting the the components from the labelled binary image and finding the bitmap of the bounding rectangle,

5. module for merging(if necessary) the broken components interactively (i.e. the horizontal and vertical heuristics can be fixed) accordingly,

6. module for multiphase character recognition scheme

7. module for translation scheme that includes insertion and traversal,

and finally interactive menu to coordinate these modules.

5.2 Possible scope of failure and their cause :

As discussed earlier,our database consists of limited number of sysmbols. Hence outside this range any symbolare not recognized.

Recognition scheme is tried to be made as simple as possible. If however two distinct characters by any chance become a single connected component then the system fails to recognize both of them. At the same time, it may recognize a character as a differnt one if confidence level matches before (eg.

\coprod will always be recognized as \prod as their bitmap differnce is very small). But again this problem can be solved by using a highly interactive menu based program and giving options for choosing thresh hold value for the confidence level.

Most vulnerable part is the insertion routine of the translation scheme. Otherwise the scheme is theoretically very strong. This part depends very much on the distance between the nearby symbols. There are every possibility that their proper relation may not be grasped due to some inherent conflicting distance heuristic values. Also these values will differ from document to document. Solution to this problem is a highly interactive menu based program where these values can be manipulated to find out the best possible solution. If however the tree is constructed properly then traversal will always give the correct result.

Future Extension :

a> Increase the template database

b> This program fails if there exists a fraction in the input file. It needs to be taken care of.

c> Make this program a highly interactive X-Windows based menu driven program

d> Test on huge number of sample bitmap images to find out the best average heuristics

1 References

1. Fundamental of Digital Image Processing.

Anil K. Jain

2. Digital Image Processing

R.C. Gonzalez & F.C. Wintz

3. Digital Picture Processing vol. 1 and vol. 2

Rosenfeld and Kak

4. Computer Vision and its application

Ramesh Gaonkar