ELSEVIER

Pattern Recognition Letters 18 (1997) 289–298

Pattern Recognition Letters

Design of a mathematical expression understanding system

Hsi-Jian Lee *, Jiumn-Shine Wang

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC

Received 1 March 1996; revised 31 December 1996

Abstract

A scientific document usually consists of text and mathematical expressions. In this paper, we present a system for segmenting and understanding text and mathematical expressions in a document. The system can be divided into six stages: page segmentation and labeling, character segmentation, feature extraction, character recognition, expression formation, and error correction and expression extraction. After we extract all text lines in a document, we separate all symbols in each text line and calculate direction-feature vectors and aspect ratios for those symbols. Then, a nearest-neighbor algorithm recognizes characters. In the expression formation stage, we build a symbol relation tree for each text line that represents the relationships among the symbols in the text line. Each text line is decomposed into a collection of primitive tokens: operands, operators and separators. Heuristic rules based on these primitive tokens are used to correct text recognition errors. Finally, we extract all mathematical expressions according to basic expression forms. Several pages of documents were scanned to test the method. All mathematical expressions are understood. In the expressions generated, a few symbols are misrecognized. The average recognition rate was 96.16%. © 1997 Elsevier Science B.V.

Keywords: Character segmentation; Character recognition; Expression formation; Error correction

1. Introduction

Two kinds of analysis are necessary to read scientific documents: page layout analysis and optical character recognition. A document may consist of various kinds of components, such as text, images, graphics, and mathematical expressions. Page layout analysis is performed to determine the physical structure of a document (Tsujimoto and Asada, 1992). Then character recognition is performed to identify characters in text components. Most proposed systems cannot handle all components of documents such as mathematical expressions and tables. In this paper, we present a system that extends the capacity of an optical character recognition system to the point of being able to understand mathematical equations.

Recently several researchers have proposed algorithms for recognizing mathematical expressions (Chen and Yin, 1992; Okamoto and Miao, 1990; Lee and Lee, 1994). They showed some sound recognition results, but several problems still exist in their recognition systems, including how to extract mathematical expressions from documents automatically, how to improve the recognition rate when various typefaces are used, and how to correct the recognition errors in mathematical expressions. This paper is intended to solve some of these problems.

* Corresponding author. E-mail: hjlee@csie.nctu.edu.tw.

0167-8655/97/$17.00 © 1997 Elsevier Science B.V. All rights reserved.
PII S0167-8655(96)00008-1

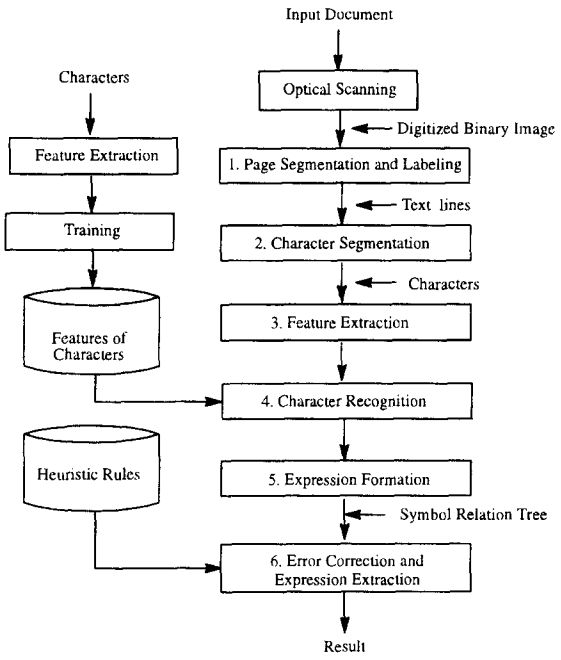

Fig. 1. System flow diagram for mathematical equation understanding.

In this paper, we propose an off-line system for understanding printed documents. Our system automatically extracts and interprets the mathematical expressions in a digitized document. The results can then be sent to a desktop publishing system that can then reconstruct the mathematical expressions. The system block diagram is shown in Fig. 1. The major components are numbered. These modules are explained in detail in the following sections.

2. Page segmentation and labeling

2.1. Text line extraction

Text-line extraction consists of segment extraction, noise removal, and segment merging. We define a segment as a rectangular area that contains a text line or a part of one. Run-length image representation is used for page segmentation because of its efficiency.

In the segment extraction process, we perform run-smearing line-by-line and extract adjacent black runs as segments. The process scans the input image from top to bottom, left to right. Since many small

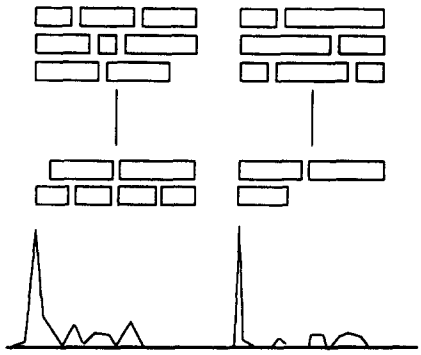

Fig. 2. Determination of leftmost column positions. The left edges of segments denote possible column locations.

segments are extracted, erroneous segment merging may result. We merge very small segments such as dots over the letters "i" and "j" into the segment nearest to them.

In the last step, we merge segments into text lines. Because a document may have multiple columns, these columns must be extracted first. Here, columns are extracted by first detecting their left edges. The leftmost column positions are defined by local maxima in a vertical projection profile (see Fig. 2). After the columns have been extracted, we extract text lines from each column. First, a horizontal projection profile is computed for all segments in the same column. We partition columns horizontally where the projection value is zero. Second, we merge neighboring segments in the same partition columns into text lines if the spaces between them are smaller than a threshold proportional to the word heights. Third, we merge neighboring text lines in different columns if the blank spaces between them are small.

2.2. Text line labeling

There are two categories of mathematical expressions in a document: *embedded* and *isolated*. An

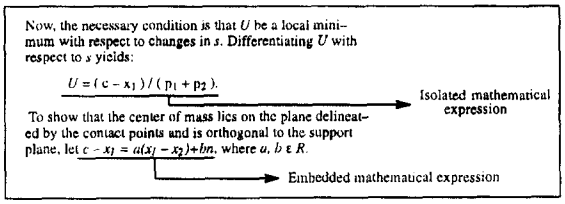

Fig. 3. An example of isolated and embedded mathematical expressions.

APPENDIX

In this appendix, it is shown that for stable poses with two points of contact, the center of mass must lie in the plane that includes these points and the normal to the supporting plane. This formed part of the necessary conditions of Section 3.2.

Consider the case of stable support by two edge points. Without loss of generality, the edges are parameterized by arc length and given by $\mathbf{x}_1(s_1)$ and $\mathbf{x}_2(s_2)$. The tangents are then $\mathbf{t}_i(s_i) = \mathbf{x}_i'(s_i)$. Contact with a plane implies that the tangents and the line between the two contact points are coplanar, or

$$(\mathbf{x}_1 - \mathbf{x}_2) \cdot (\mathbf{t}_1 \times \mathbf{t}_2) = 0. \tag{22}$$

This relates s_1 to s_2, and the partial derivatives are used to find $\partial s_2/\partial s_1$, namely

$$(\mathbf{x}_1 - \mathbf{x}_2)\left[k_1\mathbf{n}_1 \times \mathbf{t}_2 + k_2 \frac{\partial s_2}{\partial s_1}\mathbf{t}_2 \times \mathbf{n}_2\right] = 0, \tag{23}$$

where $\mathbf{n}(u, v)$ is the surface normal. The object is free to rock while maintaining two points of contact, and this corresponds to a curve of contact points given implicitly by (26) and $\mathbf{n}_1 \times \mathbf{n}_2 = 0$ for a second surface point. This curve can be represented in the domain of the surface as $[u, v] = [u(s), v(s)]$. Differentiating (26) with respect to s gives

$$(\mathbf{x}_1(u, v) - \mathbf{x}_2) \cdot (\mathbf{n}_u u' + \mathbf{n}_v v') = 0, \tag{27}$$

where $\mathbf{n}_u$ means $\partial\mathbf{n}/\partial u$. Note that this expression holds whether $\mathbf{x}_2$ is a surface or edge point. The potential energy is given by

$$U = (\mathbf{c} - \mathbf{x}_1(u, v)) \cdot \mathbf{n}(u, v). \tag{28}$$

Now, the necessary condition is that U be a local minimum for changes in s. Differentiating U with respect to s yields

$$U' = (\mathbf{c} - \mathbf{x}_1) \cdot (\mathbf{n}_u u' + \mathbf{n}_v v'), \tag{29}$$

Fig. 4. Original document image.

embedded mathematical expression is contained within a text line, while an isolated mathematical expression appears alone on a separate text line. In general, isolated mathematical expressions are taller, and the line spaces above and below them are larger than those between text lines that contain no mathematical expressions. On the basis of these two properties, text lines in a document are labeled either *TEXT* or *EXP* to denote text and expressions, respectively. Fig. 3 shows a portion of a document

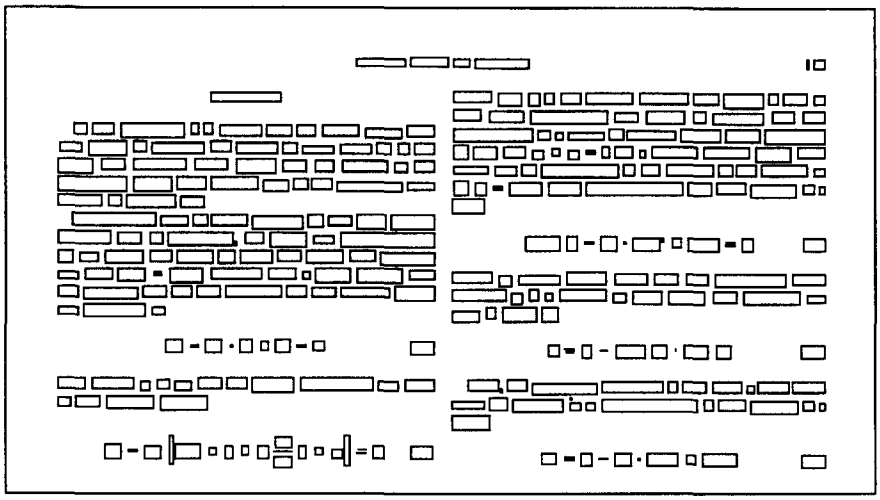

Fig. 5. Results after segment extraction and noise removal.

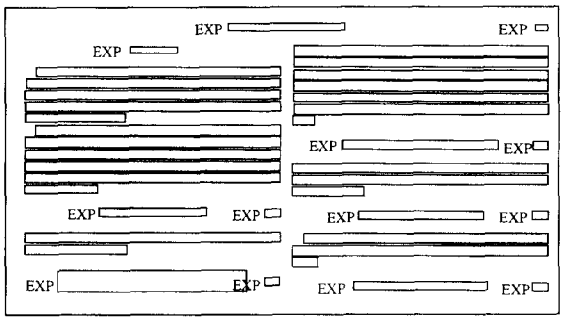

Fig. 6. Text line extraction result. Neighboring segments are merged into text lines after segment merging.

page containing isolated and embedded mathematical equations.

We use the Bayes decision rules to assign all text lines to one of two classes: text and expression, and to assign all line spacing to one of two classes: text-space and expression-space. A text line is labeled *TEXT*, if it belongs to the *text* class and the line spaces above and below belong to the *text-space* class. Otherwise it is labeled *EXP*. After text line labeling, isolated mathematical expressions can only exist on the text lines labeled *EXP*, and embedded mathematical expressions can only exist within the text lines labeled *TEXT*. Fig. 4 shows a portion of a digitized document containing mathematical expressions. Fig. 5 shows the results of segment extraction and small segment merging. Fig. 6 shows the results of text line extraction, in which horizontal neighboring segments have been merged into text lines.

3. Feature extraction and character recognition

We must separate characters in text lines, before performing feature extraction. After scanning the image of a text line once, we locate all connected components in the image and treat them as characters. We choose the directional features to represent characters by dividing characters into four rows and four columns of equally sized rectangular blocks, which results in 4×4 blocks. We next compute 4-dimensional directional features for each block; it forms a 64-dimensional feature vector (Tung and Lee, 1994).

Let W be the width and H be the height of the bounding rectangle of a character. Since the directional feature vectors are not invariant to scaling, they are normalized with respect to $(W+H)$. Although the directional feature vectors are powerful to printed characters, some problems still remain. When the aspect ratios of characters such as "$-$" (minus), "[" (left matrix symbol), "]" (right matrix symbol), and "|" (absolute symbol) are very large or very small, segmentation of these characters is not reliable. We therefore modify segmentation according to aspect ratio. When the aspect ratio $R = W/H$ of an input symbol is very large, the input image is divided into 2×5 image blocks. When the aspect ratio is very small, the input image is divided into 5×2 image blocks. Therefore, the characters are divided into three classes, denoted as Ω_1, Ω_2 and Ω_3, according to the aspect ratios.

Let (v_1, v_2, v_3, v_4) be a 4-dimensional directional feature vector extracted from each image block and normalized, where v_1 is the vertical feature, v_2 is the horizontal feature and the others are two diagonal features. When the aspect ratio of a symbol is very small (smaller than a threshold T_1), the horizontal feature and the two diagonal features are more important than the vertical feature. Therefore, we replace the directional feature vector (v_1, v_2, v_3, v_4) by (w_1, w_2, w_3, w_4) according to the following formulas:

$$w_1 = v_1, \qquad w_2 = \frac{H \times T_1}{W} \times v_2,$$

$$w_3 = \sqrt{\frac{H \times T_1}{W}} \times v_3, \qquad w_4 = \sqrt{\frac{H \times T_1}{W}} \times v_4,$$

where T_1 denotes an aspect ratio threshold value. A 40-dimensional $(5 \times 2 \times 4)$ modified directional feature vector is thus extracted. When the aspect ratio of a symbol is very large (larger than a threshold T_2), the directional feature vectors are modified similarly.

To recognize the characters, we determine the class i to which a character Y belongs by using the aspect ratio R. We next classify each input character by a minimum distance classifier using the directional features. The symbol Y belongs to the class j if

$$d_j(Y) = \min_i \left(\sum_{k=1}^{N} | y_k - m_{ik} | \right),$$

where N is the dimension of the feature vector, and m_{ik} is the mean of the kth directional features of the

ith reference pattern. The reference character set is given in Table 1.

4. Expression formation

Expression formation is used to translate the recognition results to a tree structure, called *symbol relation tree* that represents the relationships among the symbols in a mathematical expression. Six mathematical relations are taken into account: *up*, *down*, *right*, *superscript*, *subscript* and *subexpression*. They are illustrated in Fig. 7. A sample symbol relation tree is shown in Fig. 8. To build symbol relation trees for mathematical expressions, we use six spatial coordinates *min_x*, *center_x*, *max_x*, *min_y*, *center_y* and *max_y* to describe the relationships among input symbols.

The expression formation algorithm uses the concept of symbol grouping proposed by Lee and Lee (1994). A symbol group is defined as a set of symbols that are bounded by a mathematical symbol. If we represent a symbol as a function f, then its operands are bounded by the function. For example, $\sum_{i=1}^{n}$ is a *symbol group* since the two operands *from*, "$i = 1$", and *to*, "n", are bounded by the function $\sum$; x/y is a symbol group bounded by "—"; $\sqrt{x^2 + y^2}$ is a symbol group bounded by $\sqrt{}$.

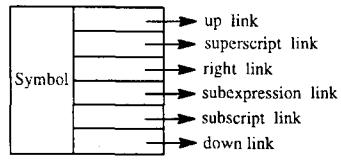

Fig. 7. Six mathematical relations.

Our expression formation algorithm operates with a list of symbols, and constructs a *symbol relation tree* that reflects the structure of the expression. The expression formation algorithm consists of three main steps as described below.

Step 1. Analyze and group characters

Each character in a mathematical expression is inspected to determine whether or not the character is a special mathematical symbol. If such a symbol is found, we generate a symbol group based on this symbol. This step is executed repeatedly to build the symbol relation tree for the symbol group. Special mathematical symbols include horizontal lines "—", summation "Σ", product "Π", integration "$\int$" and root "$\sqrt{}$".

Step 2. Form matrix

Since matrix structures cannot be detected by the previous step, we must analyze the ordered symbol

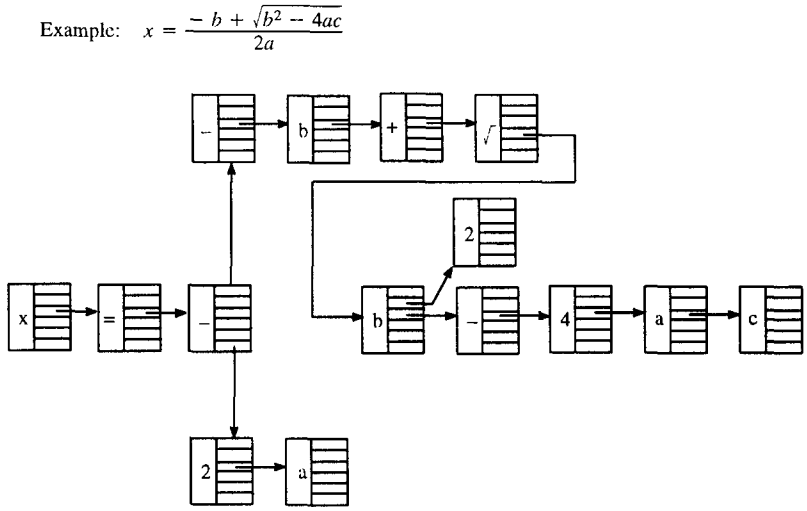

Fig. 8. An example for a symbol relation tree.

a^x x_n A^+ b_2 (a) B_* $\in^3$ $\&_a$ $X^{\&}$ (b)

Fig. 9. Several examples of superscripts and subscripts: (a) legal examples; (b) illegal examples.

list to extract matrix structures. The process of matrix formation is performed as follows:

- Find a pair of enclosure symbols (i.e., '[' and ']').
- Obtain the rectangular area enclosed by this pair.
- Find those symbols located within the rectangular area enclosed by the symbols.
- Divide these symbols into groups according to their spatial proximity. If one symbol is close to another symbol, the two are put in the same group.
- Build the matrix symbol relation tree.

Step 3. Form superscript and subscript

The superscripts or subscripts of a symbol X will be the successor of X in an ordered symbol list. We can determine the superscript or subscript relationship between two successive symbols or symbol groups by using spatial coordinates and contextual information. The symbols that can be recognized by our system are grouped into the six character classes shown in Table 1.

Table 1
Contextual character classes

Class	Description	Examples
1	Ascender	All capital letters, numerals, and b, d, f, h, k, l, i, t, b, d, θ
2	Centered	a c e m n o r s u v w x z *a c e m n o r s u v w x z* a ω μ ν σ τ π ε
3	Descender	g p q y *g p q y* γ η ψ
4	Full-height	j *j* f ξ
5	Centered	− + * #,
6	Centered	. / ? ! ^ | \ & $ < > $\in$ $\supset$ $\subset$ $\cap$ $\cup$ $\Leftarrow$ $\Rightarrow$ $\neq$ $\Leftrightarrow$ $\rightarrow$ $\leftarrow$ $\leftrightarrow$ ∇ ∂ $\oplus$ $\ominus$: $\simeq$ $\geq$ $\leq$ $\subseteq$ $\supseteq$

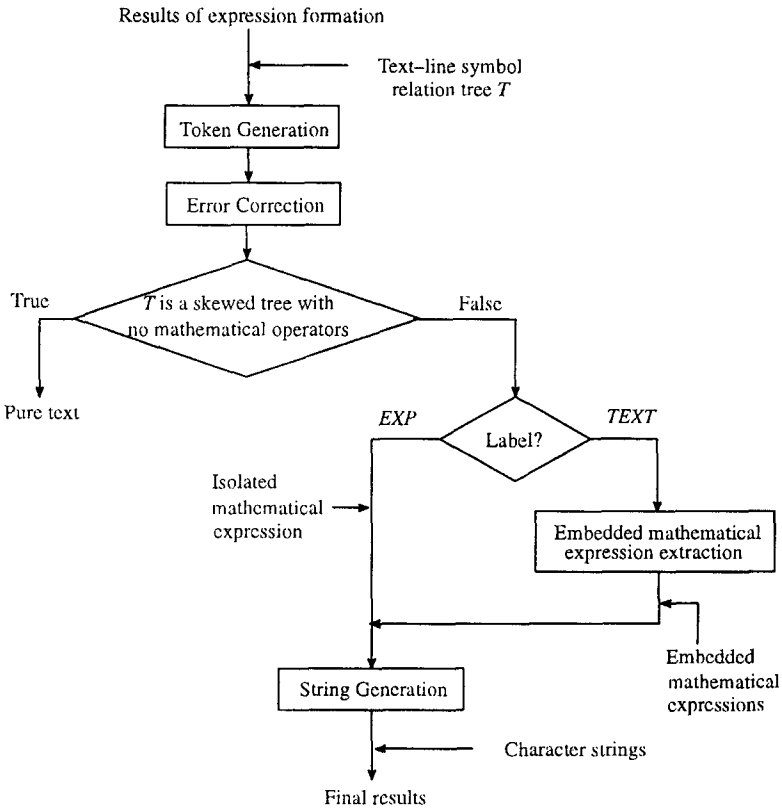

Fig. 10. Postprocessing flow diagram.

Symbols belonging to the first four classes can be superscripts or subscripts of other symbols. They can also have other symbols as superscripts or subscripts. Symbols belonging to class 5 can only be superscripts of other symbols, but cannot have other symbols as superscripts or subscripts. Symbols belonging to class 6 cannot be superscripts or subscripts of other symbols, and cannot have other symbols as superscripts or subscripts. Some legal examples are given in Fig. 9(a) and some illegal examples in Fig. 9(b). This heuristic information is used to prevent generation of illegal superscripts and subscripts.

5. Error correction and expression extraction

After expression formation, text-line symbol relation trees are built. The next task our system performs is the correction of recognition errors which may have occurred and the extraction of mathematical expressions that appear in text lines. Two postprocessing modules, an error corrector and an expression extractor, are designed. The postprocessing procedure is shown in Fig. 10, and the main steps are explained in the following sections.

5.1. Token generation

Basically, a text line is composed of operands, mathematical operators and separators. An operand is defined as a character string composed of letters and numerals. A separator is defined as a symbol that separates operands and mathematical operators within text lines. The token generation process is used to extract the primitive tokens from text lines. For this, text lines can be treated like mathematical expressions. We can traverse the symbol relation tree to extract all primitive tokens from text lines.

First, we find all linearly linked lists, whose nodes are linked in the symbol relation tree. Second, for each linearly linked list, we insert a blank separator between the two succeeding symbols if the space between them is larger than a threshold value (1 mm). Third, we scan each linearly linked list from left to right and extract all primitive tokens from it.

5.2. Error correction

At the recognition stage, there may still be some similar symbols that can result in recognition errors by our system. These similar symbols are grouped into several confusion sets, such as {V, v, *V*, *v*}, {O, o, *O*, *o*, 0 (zero)}, {S, s, *S*, *s*, 5 (five), *5* (five)}, among others. We use heuristic rules to correct the recognition errors in expressions. We check each symbol in an operand and if we find similar symbols in the operand, we can select the correct symbol from its corresponding confusion set by using the heuristic rules we propose. Below, we provide our heuristic rules and give some examples to illustrate how errors are corrected.

Rule 1. Some special function names appear frequently in expressions, such as "sin", "cos", "tan", "csc", "sec", "cot", "log" and "exp", etc. For example, in the expressions (1) $x = \tilde{5}in\ \theta$, (2) $y = c\tilde{0}sx$, the first confusion is caused by the symbols "$\tilde{5}$" and "s"; the second by the symbols "o" and "$\tilde{0}$" (zero). These recognition errors can be corrected by using this rule.

Rule 2. For every binary operator P, there must exist two operands. These two operands will generally be of the same typeface and size. See the following three expressions: (1) $B = \sum_{\tilde{/} < i < n} b_i$, (2) $A = a\tilde{X}b + c$, (3) $X = \tilde{c} + D$. The first confusion is caused by the symbols "$\tilde{1}$" and "/"; the second by the symbols "X" and "×"; the last by the symbols "c" and "C." These recognition ambiguities are flagged when rule 2 is applied.

Rule 3. There are no symbols in the subscript position of a numerical. For example, see expressions: (1) $A = \underset{\sim}{1}_2 + 3$, (2) $X = \underset{\sim}{5}_2 + a$. The first confusion is caused by the symbols "$\tilde{l}$" (ell) and "1"; the second by the symbols "S" and "5." Rule 3 changes the recognition results into "l" and "S", respectively.

Rule 4. Symbols in the same operand generally possess the same properties. See expressions: (1) $A = \underset{\sim}{x}\, y + 5$, (2) $y = 3\tilde{P}qr$. The first confusion is caused by the symbols "x" and "x"; the second by the symbols "p" and "P". Rule 4 corrects these recognition errors.

5.3. Embedded mathematical expression extraction

The final step of our system consists of an algorithm for extracting embedded mathematical expres-

Table 2
Basic forms of expressions

Form	Format	Description	Examples
1	*op_1 operator op_2*	*Operator* is a binary operator.	$a+b$
2	$op\_1_{sb}^{sp}$	*sp* and *sb* can be nil, but they cannot be null both.	C_2, xyz^4
3	symbol group	A symbol group can be an expression.	$\frac{a+b}{x}$, $\Sigma_{i=1}^{n}$, $\sqrt{a+b}$
4	Σop_1 or $\prod$ *op_1* or $\int$ *op_1*	The operators "Σ" "$\prod$" and "$\int$" have only one operand.	$\Sigma_{i=1}3i$, $\prod_{i=1}i^2$, $\int_{x=1}^{10}2x\,dx$
5	(*op_1*)	The pair enclosure symbols "(" and ")" can be replaced by "{" "," "|" and "[", and *op_1* must have been marked.	$(a+b)$, (x/y) Note: "$a+b$" and "x/y" have been marked by form 1.
6	*let_0* (*let_1*, *let_2*,..*let_n*)	The form represents a function.	$g(x, y, z)$, $f(p, q, r)$
7	*op_1*(*op_2*) or (*op_1*)*op_2* or (*op_1*)(*op_2*)	Note that these operands between the pair of enclosure symbols must have been marked.	$3xy(a+b)$
8	*operator op_1*	*Operator* has only one operand at its right.	∂x, ∇x, -3, $+5$

sions. This algorithm is divided into equation marking and equation extraction.

In equation marking, we scan all primitive tokens on text lines from left to right and mark those that belong to embedded mathematical expressions. We determine whether a primitive token belongs to an embedded mathematical expression according to the basic expression forms. The marking phase is performed iteratively until there are no primitive tokens left unmarked. The basic expression forms are listed in Table 2, where *op_k* is the *k*th operand of a basic expressions, *sb* is the subscript of an operand, *sp* is the superscript of an operand and *let_k* represents the *k*th letter of a basic expression. According to the basic expression forms, we can mark primitive tokens that form basic expressions and group them into new operands.

In equation extraction, we extract all embedded

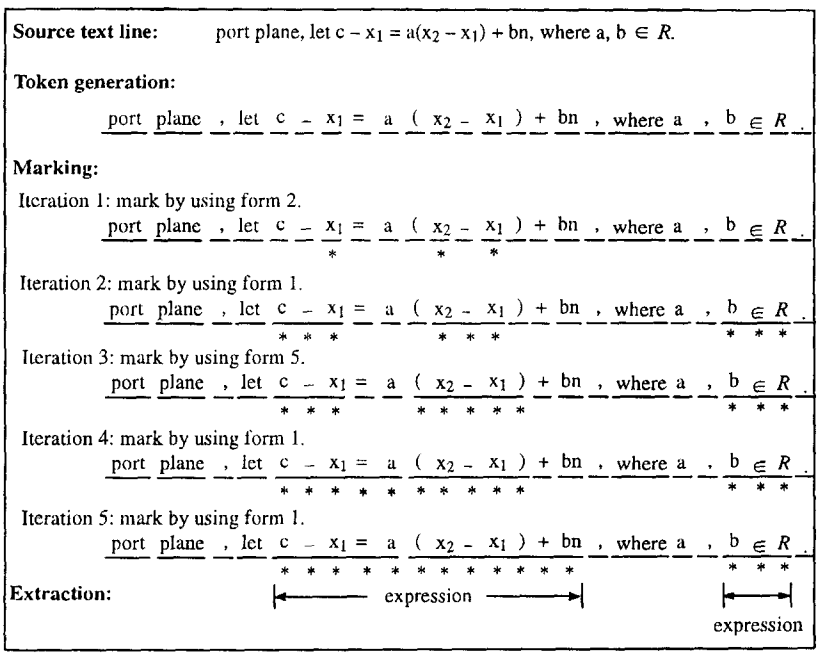

Fig. 11. An example of expression extraction.

Table 3
The result of experiments

	Document 1		Document 2		Document 3	
	Before error correction	After error correction	Before error correction	After error correction	Before correction	After error correction
No. of test symbols	3473	3473	4197	4197	3540	3540
No. of error symbols	215	137	204	147	240	144
Recognition rate for whole system	93.80%	96.06%	95.14%	96.50%	93.22%	95.93%

mathematical expressions from a text line. The task can easily be performed by scanning the primitive tokens on text lines from left to right and extracting all marked tokens in order. These tokens form embedded mathematical expressions. An example illustrating how expressions are extracted from text lines is shown in Fig. 11. Two embedded expressions are extracted from a text line in this example.

6. Experimental results

The proposed system was implemented in UNIX-C on a Sparc workstation. The original documents were scanned using Microtek MSF-300G scanner linked to a personal computer. Our symbol set included 127 letters, 36 mathematical operators, 20 numerals and 7 separators. In our experiment, we used several documents to test our system. Table 3 shows the matching results. In these experiments, the percentage of symbols in mathematical expressions is about 15%. All isolated mathematical expressions in these documents are extracted, but several embedded mathematical expressions are not identified. The experiments yielded a final recognition rate of about 96% and the error correction modules improved the recognition rate by about 2.5%.

Table 4 shows the error distribution. The major errors were due to symbol similarities, which can be reduced significantly by the process of error correction.

7. Conclusion

In this paper, we have presented a system for segmenting and interpreting text and mathematical

Table 4
Experiment error distributions

	Document 1				Document 2				Document 3			
	Before error correction		After error correction		Before error correction		After error correction		Before error correction		After error correction	
	No.	%	No.	%	No.	%	No.	%	No.	%	No.	%
Errors due to scanning	36	16.7%	36	26.3%	17	8.3%	17	11.6%	19	7.9%	19	13.2%
Errors due to symbol similarity	117	54.4%	39	28.5%	78	38.2%	21	14.3%	117	48.8%	21	14.6%
Errors due to touching symbols	26	12.1%	26	19.0%	48	23.5%	48	32.7%	67	27.9%	67	46.5%
Errors due to broken symbols	15	7.0%	15	11.0%	46	22.5%	46	31.3%	7	2.9%	7	4.9%
Misc. errors	21	9.8%	21	15.3%	15	7.4%	15	10.2%	30	12.5%	30	20.8%
Total	215	100%	137	100%	204	100%	147	100%	240	100%	144	100%

expressions in documents. The system is divided into six stages: page segmentation and labeling, character segmentation, feature extraction, character recognition, expression formation, and error correction and expression extraction. Heuristic rules are used to correct recognition errors. The recognition results demonstrate the applicability of the proposed system.

Below, we list some approaches to further improving the performance of our system.

1. Correct recognition errors due to touching and broken symbols.
2. Add more heuristic rules for correcting errors in mathematical expressions.
3. Add a contextual analysis postprocessor to correct errors in text.
4. Add a parser to check the correctness of mathematical expressions we recognize.

References

Chen, L.H. and P.Y. Yin (1992). A system for on-line recognition of handwritten mathematical expression. *Computer Processing of Chinese and Oriental Languages* 6, 19–39.

Lee, H.J. and M.J. Lee (1994). Understanding mathematical expressions using procedure-oriented transformation. *Pattern Recognition* 27, 447–457.

Okamoto, M. and B. Miao (1990). Recognition of mathematical expressions by using the layout structures of symbols. Technical Report, Institute of Information Engineering, Shinshu University.

Tsujimoto, S. and H. Asada (1992). Major components of a complete text reading system. *Proc. IEEE* 80, 1133–1149.

Tung, C.H. and H.J. Lee (1994). Increasing character recognition accuracy by detection and correction of erroneously identified characters. *Pattern Recognition* 27, 1259–1266.