A Knowledge-Based Design for Structural Analysis of Printed Mathematical Expressions

Pavan Kumar P.*, Arun Agarwal, and Chakravarthy Bhagvati

School of Computer and Information Sciences
University of Hyderabad, Hyderabad 500 046, India
pavan.ppkumar@gmail.com, {aruncs,chakcs}@uohyd.ernet.in

Abstract. Recognition of Mathematical Expressions (MEs) is a challenging Artificial Intelligence problem as MEs have a complex two dimensional structure. ME recognition involves two stages: Symbol recognition and Structural Analysis. Symbols are recognized in the first stage and spatial relationships like superscript, subscript etc., are determined in the second stage. In this paper, we have focused on structural analysis of printed MEs. For structural analysis, we have proposed a novel ternary tree based representation that captures spatial relationships among the symbols in a given ME. Proposed tree structure has been used for validation of generated ME structure. Structure validation process detects errors based on domain knowledge (mathematics) and the error feedback is used to correct the structure. Therefore, our validation process incorporates an intelligent mechanism to automatically detect and correct the errors. Proposed approach has been tested on an image database of 829 MEs collected from various mathematical documents and experimental results are reported on them.

Keywords: Mathematical expressions, structural analysis, ternary tree representation, domain knowledge, structure validation.

1 Introduction

Mathematical Expressions (MEs) form a significant part in scientific and engineering disciplines. MEs can be offline or online. Offline or Printed MEs take the form of scanned images while online MEs are written using data tablets. ME recognition is the process of converting printed or online MEs into some editable format like LaTeX, MathML etc. It is needed for applications like digitizing scientific documents, generating braille script for visually impaired etc. ME recognition involves two stages: Symbol recognition and Structural analysis. Symbols are recognized in first stage and structure (spatial relationships) is interpreted in second stage. As mentioned in [3], structural analysis plays a vital role in the overall ME recognition task.

In [11], we have discussed that mathematical symbols can be composed of one or more indivisible units called *Connected Components (CCs)*. For example,

* Corresponding author.

M.N. Murty et al. (Eds.): MIWAI 2014, LNAI 8875, pp. 112–123, 2014.

= is composed of two horizontal line CCs. For some symbols, identities depend on context. For example, a horizontal line CC can be MINUS, OVERBAR, FRACTION etc., as its identity depends on neighbouring symbols (context). If it has symbols above and below, it is a FRACTION. If it has symbols only below, it is an OVERBAR and so on. Symbols that are composed of more than one CC are called *Multi-CC* symbols. Symbols whose identities depend on context are called *Context-dependent* symbols. In [11], we have shown an architecture for structural analysis of printed MEs. This design comprises three modules: Symbol formation, Structure generation and Generation of encoding form like LaTeX. Symbol formation process takes labelled CCs of an ME image as input and forms Multi-CC symbols as well as resolves the identity of Context-dependent symbols. Elements of Multi-Line MEs (MLMEs) like matrices and enumerated functions are also extracted in this module. Structure generation module takes the formed symbols, analyzes spatial relationships like superscript, subscript etc., among them and generates an intermediate tree representation for the given ME. Third module generates an encoding form like LaTeX by traversing ME tree.

In [11], we have discussed only symbol formation process. In this paper, we have discussed the other two modules. We have also added a new *structure validation* module that automatically detects and corrects the errors before encoding form is generated. The paper is organized as follows: Existing works on structural analysis are discussed in Section 2. Section 3 gives an overall design of our approach to structural analysis. Structure generation module is presented in Section 4. Section 5 presents structure validation process along with encoding form generation. Experimental results are summarized in Section 6 and the paper is concluded in Section 7.

2 Related Work

In [18], a recent survey on ME structural analysis can be found. Existing works on structural analysis along with their intermediate representations are discussed below.

2.1 Intermediate Representation

Lee et al. [8] have proposed a data structure that captures spatial relations among the symbols of an ME. Each symbol is represented by a structure that has label information which gives identity of the symbol, and six pointers. These six pointers are meant to point to six spatially related symbols. Each symbol forms a node in the tree which is formed by joining pointers of the symbols appropriately. In this structure, most of the pointers are empty and the tree is sparse. Hence the data structure is not spatially efficient as well as it cannot handle all types of MEs like MLMEs. In [14], an ME has been represented in the form of a directed graph. Each node of the graph corresponds to a symbol and a link between two nodes has the following information: Labels of node_1 and node_2 along with spatial relationship between them (one of the above mentioned regions). As only links are stored, number of links to be used depends on

number of possible relationships among the symbols of an ME, and hence not fixed and also not known a priori. In addition, it also cannot handle MLMEs. Zanibbi et al. [19] have proposed a tree structure that contains two types of nodes: Symbol nodes and Region nodes. Symbol node represents a symbol and stores its identity. Region node is a child of symbol node that represents symbols that are spatially related (excluding the horizontally adjacent relation) to symbol node and the type of spatial relation (ABOVE, BELOW, SUPERSCRIPT, SUBSCRIPT etc.) is stored in the node. The children of region node are again symbol nodes whose symbols are represented by it and all these symbols are horizontally adjacent. Tree has region and symbol nodes at alternate levels. To handle MLMEs, the authors have extended their tree structure [17] to have TABLE (to designate MLMEs), ROW (for rows) and ELEM (for elements) nodes. In this tree structure, number of children for region nodes is not known a priori as it depends on the number of symbols in that subexpression.

2.2 Structure Generation

Structure generation process analyzes spatial relations among the symbols of a given ME and generates its intermediate representation. In [6], LaTeX code is directly generated after analyzing spatial relations without using any intermediate representation. Lee et al. [8] have proposed a method in which special mathematical symbols like $\sum$, $\int$ etc., are analyzed first, then matrices are detected and finally superscripts and subscripts of remaining symbols are captured. They have detected and extracted the elements of matrices as part of structure generation. But their approach has not been discussed in a detailed manner. Tian et al. [16] have performed structure generation of offline MEs using baseline information. In [5], a network that represents spatial relations has been constructed and minimal spanning tree that corresponds to the actual structure has been obtained. Zanibbi et al. [19] have proposed to construct baseline structure tree for an ME based on reading order and operator dominance. [15] have presented a method based on a minimum spanning tree construction and symbol dominance for online handwritten MEs. Several approaches [1,6,9] have used grammars to generate structures and hence difficult to tackle erroneous ones.

3 Proposed Design to ME Structural Analysis

Proposed architecture to ME structural analysis is shown in Fig. 1. For a given input ME image, it is binarized (converted from gray scale to binary image using Otsu's method [7]), CCs are extracted [7] and labels are assigned to them. In the above process, Minimum Bounding Rectangles (MBRs) of CCs are also computed. MBR of a CC is the minimum rectangle bounding it and is represented by its top-left and bottom-right co-ordinates. As shown in Fig. 1, proposed approach starts with labelled CCs and comprises four modules namely symbol formation, structure generation also called as *ME tree* generation, structure validation and generation of encoding form like LaTeX. Structure validation module

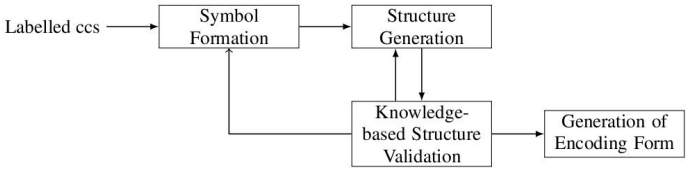

Fig. 1. Proposed architecture for ME Structural Analysis

Fig. 2. Spatial regions around a mathematical symbol **X**. TL, A, TR, BL, B and BR denote top-left, above, top-right, bottom-left, below and bottom-right regions respectively. N – Northern region, S – Southern region, E – East, W – West.

inspects tree structure and gives error feedback (shown as arrows) using domain knowledge, to symbol formation and structure generation modules. Symbol formation and structure generation processes are repeated until no errors are found by validation module, after which encoding form is generated.

As discussed in [11], symbol formation process takes labelled CCs as input and forms Multi-CC and Context-dependent symbols. MLMEs are also detected and their elements are extracted. To handle MLMEs, starting and ending delimiters like $(, [, \{, |$ and $),], \}, |$ respectively are considered as Context-dependent symbols as they can be used to enclose sub-expressions or MLMEs. Based on horizontal and vertical projection profiles [11], CCs between the delimiters are analyzed to compute rows and elements. If there is only one row and element, it is not an MLME. Otherwise, labels of the delimiters are changed to those of MLME ones and all the computed elements in each row are isolated (but their association with the delimiters is stored) from the main stream of CCs. For example, $\div$ is a Multi-CC symbol and has three CCs (one horizontal line and two DOTs). If these three CCs are vertically stacked one over another, their MBRs are combined to form a single composite symbol. Label corresponding to $\div$ is assigned to the composite symbol.

4 ME Tree Structure and Its Generation

Structure generation module takes left to right ordered symbols (from symbol formation process), analyzes spatial relations and generates an *ME tree* proposed for the purpose. In [13], we have discussed that symbols in MEs can have surrounding symbols in its top-left, above and top-right regions as well as in their bottom-left, below and bottom-right regions. For example, in $\sum_i^j$, symbols i and j are in bottom-right and top-right of $\sum$ respectively. The region

formed by combining top-left, above and top-right regions (bottom-left, below and bottom-right) in that order is called as *Northern (Southern) region*. These regions are shown in Fig. 2. Symbols in northern and southern regions of X align in its vertical direction.

Definition 1. A mathematical operator is called as *Horizontal* (H) *operator* if it does not have symbols in their northern and southern regions (Eg: +, -, $<, \leq, \geq, =$ etc.). Otherwise, it is a *Non-Horizontal* (NH) *operator* (Eg: $\sum, \int$, FRACTION etc., and accent symbols like HAT, OVERBAR etc.).

Definition 2. A symbol which is not present either in the northern or in the southern region of any other symbol in an ME is called *Baseline symbol*. For example, in $a^2 + b^2 + 2ab$, baseline symbols are: a, +, b, +, 2, a, b. Remaining two symbols 2 and 2 are in the top-right regions of a and b respectively.

4.1 Tree Representation

Proposed representation uses a ternary tree structure. Each symbol is represented using only three pointers to represent spatial relationships around it. Out of the three, one pointer is meant to point to the next baseline symbol and the other two are meant for the entire northern and southern regions respectively. In most of the MEs, the entire northern or southern region for any symbol forms a single subexpression. There may be unusual cases, where two different subexpressions are present in the same region (Eg: ${}_nC_r$). There can also be rare cases where symbols can have more than two different subexpressions (if they have pre-super and pre-sub scripts in addition to super and sub-scripts) over the northern and southern regions, but occur rarely in mathematics. Our representation handles unusual and rare cases in a different manner (discussed later).

Proposed tree node structure to represent a symbol is given by an abstract data type called as *TreeNode* with some fields. Each symbol in a given ME forms a node in the tree that is generated by linking pointers of the symbols based on their spatial relations. Each field in the data structure is discussed below:

1. Integer field, *label* gives the identity of a symbol.
2. Two boolean fields, *EOE (End of Element)* and *EOR (End of Row)* are used to handle MLMEs like matrices, enumerated functions etc. *EOE* is set to TRUE if a symbol designates end of some element of an MLME. Otherwise, it is set to FALSE. Similarly, *EOR* is set to TRUE if a symbol designates end of some row of an MLME. Otherwise, it is set to FALSE.
3. TreeNode pointer *next* of any symbol points to its next baseline symbol.
4. *nLink* of a symbol points to first baseline symbol of northern expression.
5. *sLink* of a symbol points to first baseline symbol of southern expression.
6. Northern and Southern regions for different symbols are listed below:
   (a) FRACTION – Numerator (denominator) corresponds to northern (southern) region.
   (b) SQUAREROOT – *Degree* (contained expression) corresponds to northern (southern) region.

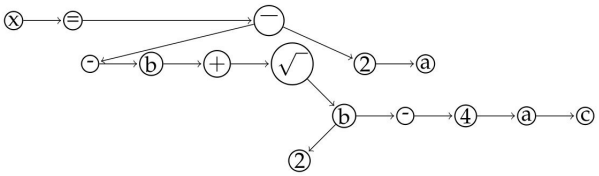

Fig. 3. Proposed tree structure for $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$. Here, x is head of the tree

(c) Accent or Wide accent symbols – Enclosed subexpression is present below (southern) for accent symbols like OVERBAR, OVERBRACE etc., and above (northern) for accent symbols like UNDERBAR, UNDERBRACE etc.
(d) Other symbols – northern (southern) region gives superscript (subscript).

In our representation, *label* information of a symbol is exploited to resolve its northern and southern regions. First node (root) of the tree (first baseline symbol) is called *head* of the tree. That means, *nLink* and *sLink* pointers point to heads of the subtrees for the northern and southern subexpressions respectively. As northern and southern regions are considered as a whole, *proposed tree structure is simple*. Proposed tree structure for an example ME is shown in Fig. 3. In this figure (and in the subsequent figures), *nLink* is shown by left link, *sLink* by right link and *next* by horizontal link.

4.2 Logical Proof for Completeness

Proposed tree structure handles unusual and rare cases in the following manner.

1. If two different sub-expressions are present in the same (northern or southern) region, one of them is logically shifted to the other region. Logically shifted regions are represented by negating the label of first baseline symbol of the corresponding region. For example, in ${}_nC_r$, if n is logically shifted, its label is negated and considered as northern child (pointed by *nLink*) of C.
2. Proposed ME representation handles rare cases using special nodes called ϵ-*nodes*. ϵ-nodes are used to handle symbols with more than two different subexpressions over their northern and southern regions. An ϵ-node also has a unique *label*, but does not refer to any symbol. Its *nLink* and *sLink* pointers can point to two more sub-expressions of a symbol, if the symbol has more than two sub-expressions around it. If a rare symbol has more than two and less than or equal to four different subexpressions around it, one ϵ-node is needed. If it has more than four different subexpressions, two ϵ-nodes are needed. ϵ-nodes are connected to the actual symbol node using *next* pointers. For example, let us consider a symbol with four sub-expressions ${}_q^p\sum_r^s$ around it. Here, super and sub scripts r and s, are pointed by $\sum$ and its pre-super and pre-sub scripts (top-left and bottom-left) p and q, are pointed by an ϵ-node. The ϵ-node is connected to $\sum$ using its *next* pointer which is shown

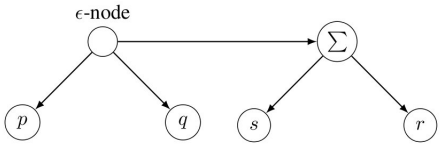

Fig. 4. Proposed tree for ${}^{p}_{q}\sum^{s}_{r}$ where $\sum$ has pre-super and sub scripts

in Fig. 4. Let us consider another rare complex example: ${}^{a}_{b}\prod^{n}_{i=1}{}^{d}_{c}$. Here, $\prod$ has six different subexpressions around it. In these cases, two ϵ-nodes are used, one to hold top-left and bottom-left subexpressions and the other one to hold above and below subexpressions. Symbol node handles the remaining two subexpressions and ϵ-nodes are connected using *next* pointers. These rare cases almost do not occur in any ME but LaTeX can generate them.

3. **Representing MLMEs:** Our proposed representation handles MLMEs by generating trees for all the elements (as each element is again an ME) recursively in all the rows and attaching them in a row-major order. That means, *next* of starting delimiter (like (, [etc.) of an MLME points to head of tree for first element in the first row. For any element in any row, *next* of its last baseline symbol points to head of tree for the next element in that row and *EOE* is set to TRUE for this last baseline symbol. For last element in any row, *next* of its last baseline symbol points to head of tree for first element of next row and *EOR* is set to TRUE for this last baseline symbol. Pointer *next* of last baseline symbol of last element in last row points to the ending delimiter (like),] etc.) if present (not present for enumerated functions). It is to be noted that the above *recursive process handles even nested MLMEs.*

4.3 Spatial Efficiency and Generality

As northern and southern subexpressions are taken as a whole, processing complexity of the proposed tree is reduced. To handle MLMEs, only two bits are used. Therefore, proposed tree is spatially efficient. In general, mathematical symbols have atmost two arguments (numerator/denominator for fraction, degree/contained expression for squareroot, super/subscripts for others etc.) and so our approach is intuitive. Proposed tree structure can also be used to handle other structures that are similar in nature to MEs. For example, chemical equations have ionic information (like oxidation state) and the number of instances of an atom in the northern and southern regions. In [12], LaTeX based linear representation has been used for ME retrieval. As proposed tree structure is simple and complete, its linear form (symbol, its northern and southern expressions in that order recursively) gives a better representation for this application.

4.4 Algorithm to Generate ME Tree

Proposed algorithm takes left to right ordered symbols of an ME as input, generates ME tree and returns a pointer to *head* of the tree:
TreeNode $*$ generateTree(L: list of Symbols)

1. Find the baseline symbols in L by isolating their northern and southern subexpressions using the next two steps. These subexpressions are first isolated for NH operators and then for other symbols (as northern and southern symbols of NH operators are present on their both sides).
2. Scan list L from left to right and if a NH operator say X, is found, do:
   (a) Create a tree node of type *TreeNode* for X.
   (b) Inspect the symbols on both sides of X in L and isolate symbols in its northern and southern regions. If a line segment obtained by joining the midpoints or centroids of MBRs of two symbols X and Y, is almost vertical (discussed earlier), then Y is in the northern or southern region of X. In our implementation, angles from 45° to 90° are considered almost vertical.
   (c) Isolated vertically aligned symbols are partitioned into northern and southern ones based on their position with respect to X. If an isolated symbol is present in the above (below) of X, then it is in the northern (southern) region of X.
   (d) If more than two subexpressions are present over the northern and southern regions, ϵ-nodes are accordingly added and those subexpressions are attached to them.
3. Scan the remaining symbols (other than NH operators and their corresponding isolated symbols in step 2) in L from left to right and for each such symbol, say X:
   (a) Create a node of type *TreeNode* for X.
   (b) Isolate symbols (using vertical alignment criteria) in its northern and southern regions by inspecting on the right side of the current symbol.
4. Connect the nodes created in steps (2) and (3) (baseline symbols) using *next* pointers. Let pointer to *head* of the tree be denoted by H.
5. Traverse through the baseline symbol nodes and for each such node, generate trees for their northern and southern subexpressions recursively:
   (a) Let pointer to the current baseline symbol node be denoted by X (Initially, $X = H$). Generate ME tree for northern region of X, if present, and assign pointer to its head to *nLink* of X. Let symbol list in the northern region (captured either in step (2) or (3) above) be denoted by NR. Therefore, $X \rightarrow nLink = generateTree(NR)$.
   (b) Similarly, $X \rightarrow sLink = generateTree(SR)$, where SR denotes symbol list in the captured southern region.
   (c) Go to the next baseline symbol node. $X = X \rightarrow next$. Go to step 5(a).
6. Traverse through baseline symbol nodes and if any MLME delimiter is found, generate and attach trees of its elements in a row-major order recursively.
7. Return *head* of the final tree generated for the given ME. *return* H.

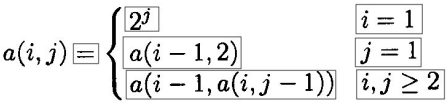

Fig. 5. An enumerated function image with its multi-CC symbols and extracted elements (shown in boxes) after symbol formation process

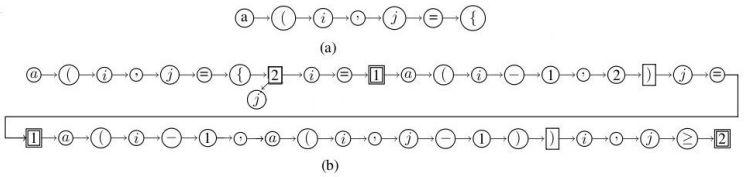

Fig. 6. ME tree generation for the enumerated function shown in Fig. 5 (a) Initial tree (b) Final tree after the elements are attached

An enumerated function image is shown in Fig. 5, in which, Multi-CC symbols are formed as well as its elements are extracted in the symbol formation process (shown in boxes). Its ME tree generation process is shown in Fig. 6, in which intial and final trees (before and after element attachment) are shown in Fig. 6(a) and (b) respectively. For rectangular nodes, *EOE* is set to TRUE and for double rectangular nodes, *EOR* is set to TRUE.

5 Structure Validation

Generated ME tree is validated using domain (mathematics) knowledge to verify correctness of the structure. If any erroneous structure is detected, corresponding feedback is given to symbol formation or structure generation modules, according to the source of error. Common errors that occur in the symbol formation module are Multi-CC as well as Context-dependent symbols may not be correctly resolved. Similarly in structure generation module, super or sub-script relationships may be lost for a symbol. Our validation algorithm is given below:

1. ME tree is traversed in pre-order [4] (visit in the following order: current node, its *nLink*, *sLink* and *next* recursively) and verified for errors using domain knowledge. If no errors are found, encoding form is directly generated. Otherwise, validation module decides if the errors are from symbol formation or structure generation modules.
2. If an error occurs in symbol formation module, feedback is given to that module and that process is repeated again by taking that error into account. Newly formed symbols are sent to structure generation module and the new tree structure generated is again validated.
3. If the error occurs in structure generation module, a new tree structure is generated by taking into account the error feedback.
4. The above two steps are repeated until no errors are found.

$$f'(a) = \left.\frac{df}{dx}\right|_{x=a} = \left.\frac{dy}{dx}\right|_{x=a} = Df(a)$$

(a)

```
f'(a)= \frac {df}{dx}
\left|\begin{array}{cc}
dy \\ - \\ - \\ - \\
x=a & dx \\ \end{array}
\right|_{x=a}=Df(a)
```

(b)

$$f'(a) = \frac{df}{dx}\begin{vmatrix} dy \\ - \\ - \\ - \\ x=a & dx \end{vmatrix}_{x=a} = Df(a)$$

(c)

```
f'(a)= \frac {df}{dx}
|_{x=a}= \frac {dy}{dx}
|_{x=a}=Df(a)
```

(d)

$$f'(a) = \frac{df}{dx}|_{x=a} = \frac{dy}{dx}|_{x=a} = Df(a)$$

(e)

Fig. 7. (a) An ME image (b) LaTeX output without validation (c) Regenerated ME from LaTeX output (b) (incorrect) (d) LaTeX output with error feedback to symbol formation process (e) Regenerated ME from LaTeX output (d) (corrected)

Validation module uses domain knowledge (mathematical properties encoded in the form of rules) to detect errors in the tree. The knowledge can always be updated over time like that for any knowledge-based system [2]. Some of the rules used are: (1) An expression (or subexpression) should not end with any operator. (2) Horizontal operators do not have superscript or subscript expressions. (3) Superscript and subscript expressions of NH operators should not have matrix-like expressions, except determinants (as determinants are logically scalars). (4) Elements of a matrix do not have operators alone or they do not have equations.

Generation of Encoding Form: After errors in the ME tree structure are rectified by validation module, encoding form like LaTeX, MathML etc., can be generated by traversing the tree. In our approach, LaTeX code is generated by maintaining a mapping table that maps a given symbol to its LaTeX encoded symbol. The algorithm to generate encoding form is based on pre-order traversal on the ME tree: (1) Inspect the label of the current node (which is head of tree initially) and generate its encoded symbol. (2) Encode its northern and southern subtrees recursively. (3) Inspect *EOR* and *EOE* fields and if they are true, add row and element delimiters (\\ and & for LaTeX) accordingly. (4) Move to the next baseline symbol and repeat the first three steps.

An ME image is shown in Fig. 7(a). For this ME, delimiters are taken as MLME ones (determinants) with five rows and two columns by the symbol formation process and its LaTeX output (without validation) is shown in Fig. 7(b). Its regenerated ME is also shown in Fig. 7(c) for better understandability. Validation module finds an equation in the first element of the last row ($x = a$) of the determinant and gives feedback to the symbol formation module that it is not a determinant. It is considered by symbol formation module and the remaining procedure is repeated to get correct tree structure. Its LaTeX output as well as its regenerated ME are shown in Figs. 7(d) and (e) respectively.

Table 1. Summary of results on our PACME database of 829 MEs

ME type	Field	Number of MEs	Without Validation		With Validation	
			Number of correctly captured structures	Accuracy (%)	Number of correctly captured structures	Accuracy (%)
Non-MLMEs	Algebra	251	189	75.3	212	84.4
	Trigonometry	116	108	93.1	112	96.6
	Geometry	17	13	76.5	17	100
	Calculus	374	321	85.8	352	94.1
MLMEs	Matrices	62	50	80.6	53	85.4
	Enumerated functions	9	7	77.8	8	88.9
Grand Total		829	688	82.9	754	90.9

6 Experimental Results and Discussion

For our experimentation, we have created a database of 829 ME images called *PACME* database [10] that are collected from various mathematical books [20]. Those MEs cover different fields of mathematics like Algebra, Trigonometry, Geometry, Calculus etc., and range from simple to complex formulae and equations. Each ME image in the database is binarized and CCs are extracted. Labels of these CCs are manually stored in a file. Our approach to ME structural analysis has been implemented in C++. As mentioned in Section 3, for a given ME image, proposed approach binarizes it, extracts the CCs and uses its corresponding file to label those CCs. After CC labelling, it performs the task of structural analysis. Results on our PACME database of 829 MEs are summarized in Table 1. In that table, percentage accuracies without and with structure validation are shown separately (to give a complete picture of our database) for each ME type (MLMEs and Non-MLMEs) and field in that type. Accuracy is the ratio of the number of correctly captured ME structures to the total number of MEs. The above experimentation shows that validation process (based on our tree structure) has given 8% improvement in accuracy.

7 Conclusions and Future Directions

In this paper, we have proposed a ternary tree based representation for structural analysis of printed MEs. We have shown that the proposed tree is simple, complete and spatially efficient. It can be used to represent other similar structures like chemical equations and also in applications like ME retrieval. Generated tree structure is validated using domain knowledge and error feedback is used to automatically correct the errors. Proposed approach that is based on an expert system incorporates an intelligent mechanism in the process. Our approach has been tested on a database of 829 ME images and experimental results are reported on them. In future, semantic knowledge can be incorporated into validation process to improve error detection and correction.

References

1. Álvaro, F., Sánchez, J.A., Benedí, J.M.: Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1012–1016 (2013)

2. Buchanan, B.G., Shortliffe, E.H.: Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project. Addison-Wesley (1984)
3. Chan, K.F., Yeung, D.Y.: An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions. Pattern Recognition 33(3), 375–384 (2000)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company (1989)
5. Eto, Y., Suzuki, M.: Mathematical formula recognition using virtual link network. In: ICDAR 2001, pp. 762–767. IEEE Computer Society, Washington, DC (2001)
6. Garain, U., Chaudhuri, B.B.: Recognition of online handwritten mathematical expressions. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34(6), 2366–2376 (2004)
7. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 2nd edn. Pearson Education Indian Reprint (2003)
8. Lee, H.J., Wang, J.S.: Design of a mathematical expression understanding system. Pattern Recognition Letters 18(3), 289–298 (1997)
9. MacLean, S., Labahn, G.: A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. IJDAR 16(2), 139–163 (2013)
10. PACME: Printed Mathematical Expression Image Database (2010), `http://dcis.uohyd.ernet.in/~pavanp/mathocr/PrintedMEs.zip`
11. Pavan Kumar, P., Agarwal, A., Bhagvati, C.: A rule-based approach to form mathematical symbols in printed mathematical expressions. In: Sombattheera, C., Agarwal, A., Udgata, S.K., Lavangnananda, K. (eds.) MIWAI 2011. LNCS (LNAI), vol. 7080, pp. 181–192. Springer, Heidelberg (2011)
12. Pavan Kumar, P., Agarwal, A., Bhagvati, C.: A structure based approach for mathematical expression retrieval. In: Sombattheera, C., Loi, N.K., Wankar, R., Quan, T. (eds.) MIWAI 2012. LNCS (LNAI), vol. 7694, pp. 23–34. Springer, Heidelberg (2012)
13. Pavan Kumar, P., Agarwal, A., Bhagvati, C.: A string matching based algorithm for performance evaluation of mathematical expression recognition. Sadhana 39(1), 63–79 (2014)
14. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., Kanahori, T.: Infty-an integrated OCR system for mathematical documents. In: Proceedings of ACM Symposium on Document Engineering 2003, pp. 95–104. ACM Press (2003)
15. Tapia, E., Rojas, R.: Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In: Lladós, J., Kwon, Y.-B. (eds.) GREC 2003. LNCS, vol. 3088, pp. 329–340. Springer, Heidelberg (2004)
16. Tian, X., Fan, H.: Structural analysis based on baseline in printed mathematical expressions. In: PDCAT 2005, pp. 787–790 (2005)
17. Zanibbi, R., Blostein, D., Cordy, J.R.: Directions in recognizing tabular structures of handwritten mathematics notation. In: Proceedings of IAPR International Workshop on Graphics Recognition (2001)
18. Zanibbi, R., Blostein, D.: Recognition and retrieval of mathematical expressions. IJDAR 15(4), 331–357 (2012)
19. Zanibbi, R., Blostein, D., Cordy, J.R.: Recognizing mathematical expressions using tree transformation. IEEE Transactions on PAMI 24(11), 1455–1467 (2002)
20. Zwillinger, D.: CRC Standard Mathematical Tables and Formulae, 30th edn. CRC Press, Boca Raton (1996)