2011 International Conference on Document Analysis and Recognition

Probabilistic Mathematical Formula Recognition Using a 2D Context-Free Graph Grammar

Mehmet Celik
Department of Computer Engineering
Bilkent University
Ankara, Turkey
Email: mcelik@cs.bilkent.edu

Berrin Yanikoglu
Faculty of Engineering and Natural Sciences
Sabanci University
Istanbul, Turkey
Email: berrin@sabanciuniv.edu

***Abstract*—We present a probabilistic framework for the mathematical expression recognition problem. The developed system is flexible in that its grammar can be extended easily thanks to its graph grammar which eliminates the need for specifying rule precedence. It is also optimal in the sense that all possible interpretations of the expressions are expanded without making early commitments or hard decisions. In this paper, we give an overview of the whole system and describe in detail the graph grammar and the parsing process used in the system, along with some preliminary results on character, structure and expression recognition performances.**

***Keywords*-online, handwriting recognition, OCR, mathematical equation, 2D-grammar, graph grammar**

I. Introduction

In spite of the ever-growing place of computers and other digital devices in our lives, pen and paper is still the most convenient way for communicating or recording information. In particular, mathematical expressions are most conveniently entered by handwriting. Computer understanding of handwritten text (handwriting or mathematical formulas) is an ongoing research area. The difficulty is due to several factors, including large variations in writing styles, the size of the lexicon indicating the possible alternatives, and ambiguity of certain shapes without semantic understanding (e.g. 'O' and '0').

Mathematical expression recognition includes two main subproblems: *character recognition* for recognizing segmented and tokenized symbols (numbers, letters, special mathematical symbols) and *structural analysis* for understanding the structure of the expression from the spatial relationships between the characters and the character recognition output. Recognizing mathematical expressions is more challenging compared to handwritten text in recognition due to the complex semantics of mathematical expressions as well as the 2-dimensional layout of the characters.

There are several approaches in literature for structural analysis of mathematical expressions: procedurally coded rules [1]; X-Y cuts based on projection profiles [2], [3]; baseline tree construction [4], [5]; stochastic context-free grammars [6]; constraint attribute grammars [7]; hierarchical decomposition parsing [8]; spanning tree generation on weighted graphs [9]; and graph grammars [10]–[13]. Among these approaches, graph grammars have certain advantages: as put forward in [14], graph grammars by their nature are 2-dimensional representations that can represent a possibly infinite number of patterns with finite number of rules, when augmented with attributes. Indeed, graph grammars are among the preferred approaches to formula recognition in recent years. In [10], a graph grammar is added to an existing system to relax their constraints about writing order of symbols. Work in [11]–[13] are based on graph re-writing, where a bottom up parser is used, collapsing matched nodes into a single node at each rule application. For these systems, the output of the parse process is a single node containing all of the input symbols and corresponding to the intended meaning of the expression.

Our system uses a probabilistic context-free graph grammar that guides the system to find mathematically valid interpretations and associate probabilities to each possible interpretation of the expression. The proposed system distinguishes itself from the previous work in its probabilistic approach: whereas previous graph-grammar based approaches modify the initial graph with the application of the chosen grammar rules irreversibly, our approach entertains all possible interpretations of neighboring tokens and eventually the expression. This is possible thanks to its graph-grammar which eliminates the need for specifying rule precedence, where all possible interpretations generated thus far are kept in an extended graph. Within this framework, the disambiguation of all possible interpretations of the expression is done at the end of the parsing, by considering the likelihoods of the resulting interpretations. The likelihood of an interpretation depends on the suitability of the symbols spatial distribution for the rules used and the and the likelihoods of the recognized symbols. The output of our system is a list of the most-likely parses of the input, along with their likelihoods. This is an important advantage of the proposed system, as the user can simply choose the correct parse from the list, rather than correcting the parse result or rewriting the expression.

1520-5363/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDAR.2011.41

The next section gives a brief description of graph grammars and then our approach and experimental results are described in subsequent sections. In the rest of the paper, we use the term OCR to shortly refer to character recognition; symbol and character interchangeably to refer segmented characters; and node or token refer to the current group of characters that form a subexpression.

II. Graph Grammars

Mathematical formulas are precise and the grammar of mathematics strictly defines what is a proper mathematical expression and the correct parse (intended meaning) of a given mathematical expression. A grammar consists of production rules that indicate how terminals and non-terminals defined in the grammar, are combined to produce non-terminals as a result of the rule application. For instance we can give a simple string grammar that defines rules that makes up digits and integers as follows (here the 10 digits are the terminals, while **digit** and **int** are the non-terminals):

$\mathbf{digit} \Rightarrow 0|1|2|3|4|5|6|7|8|9$
$\mathbf{int} \Rightarrow \{-\}\mathbf{digit}\{\mathbf{digit}\}$

Graph grammars provide a formalism for grammatical processing of multi-dimensional data which cannot be achieved by string grammars. Since their introduction to solve picture processing problems, graph grammars have been used in diverse areas such as concurrent systems, databases, programming languages and biology [15]. In mathematical expression recognition, a graph grammar is often used in conjunction with graph rewriting methods where the initial graph constructed from the tokenized expression is iteratively reduced to a single node graph corresponding to the parse tree of the expression. In each iteration, a grammar rule is selected and applied, when a subgraph of the current graph matches the pattern graph of the rule; as a result of the rule application, the current graph is transformed as indicated by the rule.

Specifically, a rule $r = (g_l, g_r, C, Em)$ consists of a left-hand side graph g_l and a right-hand side graph g_r, an applicability predicate C, and an embedding rule Em. The **applicability predicate** C is a set of constraints on attribute values of nodes and/or edges, and non-existence of certain edges, that need to be satisfied, so as to be able to apply the grammar rule. For instance, the application predicate of a rule about superscripts indicates that two neighboring tokens should have acceptable size and position relationships. With applicability predicates, the application of a production rule can be restricted even if the rule has a match in the input graph. A **production** is the application of a rule r to graph G to produce G', which is denoted as $G \Rightarrow_r G'$. With a production $G \Rightarrow_r G'$, an occurrence of a subgraph g_l in G is replaced with a subgraph g_r to produce G', according to embedding rule Em, if the applicability predicate C is satisfied. The **embedding rule** specifies how to place the subgraph g_r within the graph containing the original subgraph g_l. In string grammars, the placement of the production is obvious, but in graph grammars, placement of production graph g_r has to be specified via the embedding rule Em that describes how to handle dangling edges (edges that lose one of their nodes after the g_l is removed from the graph) and how to connect produced graph g_r to the existing graph. A graph $G = (n, e)$ is said to be in **graph grammar** GG if and only if $n \in N$ (nodes) and $e \in E$ (edges) of GG and there exists a derivation that can generate G with rules from R. Here a derivation from graph G to graph G' of grammar GG is defined as a sequence of productions where $G \Rightarrow_{r_{i1}} G_1 \Rightarrow_{r_{i2}} G_2 ... \Rightarrow_{r_{ik}} G'$. Figure 1 shows an example rule r and a derivation from graph G to G', where nodes labeled a and c are replaced with node d if there is a directed edge from a to c. The embedding rule indicates that only edges towards c and edges outgoing from a should be kept. The dashed nodes and edges in the rule r indicate possible extra nodes and edges, which may or may not be present in the actual

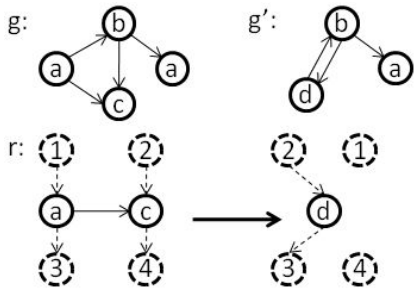

Figure 1. Application of rule r to graph g gives graph g'.

III. Proposed Method

An overview of the proposed system is illustrated in Fig 2. The input expression is first segmented into isolated symbols (a character or a stroke of a character) and each symbol is recognized by the OCR engine explained in Section III-A. Then an initial graph is constructed where the nodes represent the recognized symbols and edges represent the detected spatial neighborhood between symbols, as explained in Section III-B.

The parse algorithm applies grammar rules to the current graph, adding a new node and its edges in each iteration (see intermediate tokens in Fig. 2). These new nodes or tokens represent possible interpretations of neighboring tokens. The parse process continues until there is no valid production left.

Our grammar and the parse algorithm are explained in sections III-C and III-D.

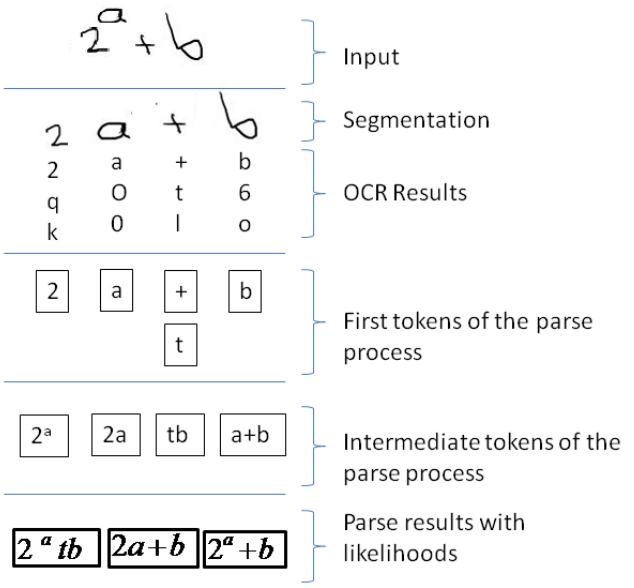

Figure 2. Overview of the parsing process. The constructed tokens correspond to the nodes in the graph while the edges are not shown for clarity.

A. Segmentation and Symbol Recognition (OCR)

The input expression is first segmented into individual characters based on time dimension, whereby a relatively large time difference between two consecutive strokes or characters is used to indicate character boundary. Then, spatially overlapping symbols are re-grouped (e.g. the two strokes of a '+' sign).

The OCR system is a combination of a Support Vector Machine (SVM) and an Artificial Neural Network (ANN), used in conjunction. It takes as input the segmented characters and outputs top-3 alternatives with associated confidence scores. We have selected a subset from the LaViola data set [5], shown in the Appendix.

Preprocessing consists of size normalization which is done on the online data to reduce artifacts whereby the coordinate of each point is mapped into a fixed coordinate range. Then a character image is created from these points by interpolation. Feature extraction takes as input the image of the resized character, ignoring time dimension. This is done to eliminate temporal variations in the drawing of characters, as well as allowing user corrections of symbols and formula that may be done after the equation is completed. For both classifiers, the input features consist of horizontal, vertical and diagonal histograms of the symbol images; horizontal, vertical and diagonal depths of the first black pixels of the symbol images; number of black pixels in a 8 by 8 windows over the whole symbol image; and ratio of width to height.

The success rate of the SVM system on this data is 92%. Although there are methods to generate posterior probabilities from multi-class SVM classification, we used an ANN to generate classification alternatives and obtain reliable recognition confidence. The ANN classifier we used is a 1-hidden layer feedforward neural network with 30 hidden neurons. The performance of this classifier is lower compared to the SVM, with top-1 and top3 recognition rates of 88% and 97%, respectively. Since the SVM is more successful in the top-1 performance, the OCR system uses the SVM output as the top-choice and gets the next two choices and the confidences from the ANN. While the accuracy is lower than state-of-the-art results, OCR was not the main focus in this work.

B. Constructing the Initial Graph

The initial graph is generated from a list of tokens obtained by the segmentation and passed through the OCR engine. In this graph, a node correspond to a token and an edge between two nodes indicate that these two nodes are neighbors in the spatial layout of the expression. The process can be explained precisely using the following definitions of the graph elements:

Nodes: A node is a tuple $n = (t, i, c, A)$ where t is the type of the node; i is a unique identifier; c is the identifier of the rule that constructed the node; and A is a set of attribute values. The type of a node t is the lexical type of the symbols, such as number, letter, operator. Each node knows which rule constructed itself, so if needed, the whole history can be generated. Each box in Fig. 4 represents a node in the graph.

Edges: An edge is tuple $e = (t, n_1, n_2)$ where t is the type of the edge, n_1 and n_2 are nodes that are connected together by the edge. There are three types of edges used in the parse process:

- **Spatial relationship edges** indicate if two nodes are *neighbors* (see definition below).
- **Component edges** between a non-terminal node and its components, are used to generate the syntax tree after the parse process.
- **Production edges** are the reverse of component edges, linking a terminal or non-terminal node to a non-terminal node that is produced using it.

The initial graph only have spatial relationship edges and they determine the outcome, while others (component and production edges) are used to keep track and speed-up the parse process.

In the proposed system, spatial relationship edges do not have any attributes since we do not distinguish between different types of neighborhood relationship (side, top, bottom etc.); different neighborhood types are *implicitly* decided by each rule's applicability predicate. The advantage of our approach is that by associating spatial relationship attributes with applicability predicates of the rules, as opposed to defining global definitions for spatial relationships, each rule can have its own definition for spatial relationships categories. In this way, rather than rigidly labeling two symbols that are written with a weak y-offset with a side

neighborhood edge, the subscript rule for instance decides if the two symbols' relative positions are sufficient to apply the rule. Neighborhood itself is defined as having a clear line of sight between the center points of their bounding boxes and being at a distance smaller than a threshold value calculated from the median size of the symbols in the expression.

Normally, a token has 3 best recognition alternatives associated with it. However as shown in Fig.2, if a character may belong to more than one type of symbol (e.g. "+" may be an operand or the symbol "t"), then 2 tokens are generated for it in order to simplify the parse process.

C. Grammar

We use a probabilistic, context-free 2D-grammar which is based on the mathematical syntax, using the spatial layout information in the applicability predicates of the rules. In this grammar, a rule is a tuple $r = (g_r, g_l, C)$ where g_l is the pattern graph, g_r is the product graph and C is the applicability predicate such that $C : g_m \rightarrow \{TRUE, FALSE\}$ where g_m is a graph matched with g_l. There is no embedding rule because all rules follow the same embedding. Normally a graph grammar rule indicates that g_l is replaced by g_r but in our system, it indicates that g_r is added to the graph (as a new node) and g_l is kept as it is.

The left-hand-side graph g_l of each rule is a **star graph** (a graph that has a central node and surrounding neighboring nodes connected only to the central node), and the right-hand-side graph g_r is a single node. Fig. 3 exemplifies g_l and g_r graphs of two simple grammar rules, where the '+' operator in rule r_1 and α in rule r_2 are the central nodes of the rules.

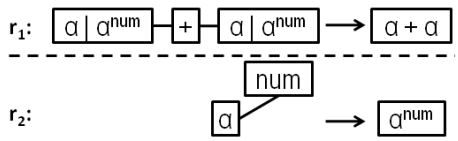

Figure 3. Sample rules where "|" depicts "or".

The most important part of the decision to apply a rule comes from applicability predicates. For most rules, the angle and distance between symbols are checked, as well as their sizes. Some rules may have further checks on attribute values. For example, for the rule that checks for fractions, g_l has a central node which represents the horizontal line symbol. The constraints used in the applicability predicates are kept loose, in order to keep all likely interpretations of the mathematical expression. For instance, the superscript rule does not require that the superscript symbol is smaller in size than the base, but that it is not (much) bigger.

Since the matched nodes are kept in the graph, each rule also has a predicate that checks the non-existence of a production edge that connects to a node which is same as g_r of the rule, to prevent matching same nodes again and generating the same product. This somewhat complicates the parse process and increases the complexity but removes the need for defining precedence rules.

There are currently 17 rules in the grammar, including mathematical rules for subscript, superscript, operator ('+','-', '$\times$', and '$\div$'), fraction, summation, and integral), as well as few rules for combining symbols written in multiple non-overlapping strokes (e.g. '=','$\div$'). Some of the terminals and non-terminals defined in the grammar is given in the Appendix.

The developed system was designed mainly to convert handwritten mathematical expressions into LaTeX for easy entry of scientific articles; as such, the LaTeX code of the correct parse is unambiguous. However, the system does not know about mathematical precedence rules, so as a result two or more likely parse alternatives would be generated for an input which would only be resolved with precedence rules (e.g. $a + b \times c + d$). However, since the system offers all likely interpretations to the user, the user may chose the right interpretation among the several likely interpretations.

D. Parse Algorithm

Our parse algorithm is a fairly straightforward bottom-up process. In each round, the algorithm checks what rule of the grammar may be applicable for each token in the graph. As illustrated in Fig. 4, there are initially 4 tokens corresponding to 4 nodes in the initial graph; then, after the first round, two new tokens (a^2 and $a+b$) are generated and added to the graph.

Specifically, two tasks have to be done by the parser: finding a match for pattern graphs of the rules and embedding the resulting product graph. Since the pattern graph of any rule is a star graph in our system, when processing a node, the parser looks for a matching rule which has the same center node; then it checks for the neighboring nodes and applicability predicates to finalize the matching process.

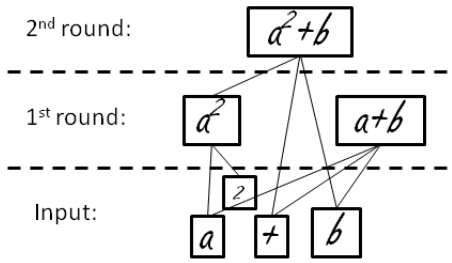

Figure 4. Nodes generated in each round, with their component edges.

Once a match is found, a new node is generated according to the rule, which then gets connected to the existing graph with component and production edges. Spatial relationship edges are generated among newly produced nodes after no possible production is left in the existing graph. Each new node inherits the neighbors of its components and

spatial relationship edges between new nodes are generated separately.

In order to deal with the complexity of the parse process where all possible local interpretations are kept, we use the fitness (likelihood) of the token to decide whether to expand that token (i.e. apply rules). Ideally this would be done with A^* search, but currently it is done by a fitness threshold which is dynamically adjusted depending on the total number of tokens and the fitness and coverage (how many tokens of the input expressions are covered) of the best token.

E. Disambiguating the Parse Results

The output of the parse process is a graph where all possible productions are present. Furthermore, if the input can be defined by the grammar, then at least one node which covers all input symbols will be in the output graph. Since component edges keep the history of productions, an expression tree can be generated if one of the alternative interpretation nodes is selected as the root and component edges are followed until it reaches a terminal node.

We compute the likelihood, also referred as fitness, of each parse alternative according to pre-learned spatial layout and OCR output probability distributions. These distributions, for instance for the relative size differences of base and subscript symbols, are learned offline from separate training data. In short, the likelihood of each generated node is the average log likelihood of the spatial relations that generated the node and the likelihoods of the components which is the probability of occurence of a symbol.

To give an example, for the input in Fig. 4, the likelihood of the token $a + b$ would depend on the likelihood of the spatial layout of the symbols 'a', 'b' and '+', with respect to the rule generating the addition. We model each spatial distribution statistics as a histogram and compute the likelihood of a given distance between two symbols (e.g. x or y offset between 'a' and '+') with respect to this histogram. We also use the character recognition probabilities to differentiate between alternative parses $a + b$ and atb which share the same layout likelihood, but differ in the likelihood of the letter 't'. The likelihood of a more complex expression (e.g. $(a+b)^2$) is computed by averagingthe log-likelihoods of its components weighted by the number of components in each component. The likelihood calculation is done at each rule application.

IV. Experimental Results

The developed system is tested using a portion of the mathematical expression database collected in association with this work [16]. The full database contains 57 equations each from 15 different users, chosen from common expressions to match the ones used by Vuong et al [17]. Expression lengths ranges from 7 to 30 characters in length. The test set consists of 20 equations, each written by 5 different users.

The results are analyzed in terms of expression recognition accuracy (the LaTeX code generated for the equation is correct); structure recognition accuracy (the LaTeX code is correct except for OCR mistakes); and character recognition accuracy, as illustrated in Table IV.

Task	**Accuracy**	**Count**
Correctly Recognized Expressions	17%	17/100
Correct Structural Analysis	50%	50/100
Correct Character Recognition	79%	1100/1410

Table I
OVERALL ACCURACY RESULTS (5 USERS X 20 EXPRESSIONS EACH)

Task	**Accuracy**	**Ratio**
Expression Length $\leq$ 10		25/100
Correctly Recognized Expressions	52%	13/25
Correct Structural Analysis	88%	22/25
Expression Length $\in [11-30]$		75/100
Correctly Recognized Expressions	5,33%	4/75
Correct Structural Analysis	37,33%	28/75

Table II
RESULTS ANALYZED IN TERMS OF THE LENGTH OF THE EXPRESSIONS.

We see that equation recognition accuracy is low (17%), which is not very suprising given the difficulty of the problem; but structure recognition accuracy is also not very high either (50%). This can be explained by the fact that both overall and structure errors are affected significantly by OCR accuracy. For instance if a parenthesis, the integral sign or the plus sign is misrecognized, the structure analysis fails. Unfortunately these special characters are among the ones that also incur the largest OCR mistakes. Indeed, character recognition accuracy for characters occuring in equations is much lower compared to isolated character accuracy developed in this work (79% vs. 91%). Table II shows the distribution of errors according to length of the expressions.

Accuracy results reported in literature for online mathematical expression recognition show a large variance (27-75% for recognition and 91-98% for structure recognition [16]). One must be careful in comparing results, as systems differ in many aspects, from the complexity of the grammars to the test database size and complexity. In addition to different databases used, one important factor that increases the accuracy among the reported results is feedback given to the user while s/he is writing; resulting in a perfect segmentation and OCR result. In a second test with 10 users and 4 expression from each, we tested this scenario by manually tokenizing the expressions and entering the correct OCR results, resulting in an 85% correct structure recognition, failing typically only for long expressions by time out.

V. Conclusion

We presented a system for mathematical expression recognition, using a 2D-graph grammar that generates all likely

parse alternatives of the expressions, along with their likelihoods. Given that mathematical expression recognition problem is far from being solved, this is an important feature since the user can select the correct alternative from this list, rather than tediously correcting the input or the output. The grammar used in the system is a star grammar that eliminates the need for specifying rule precedence by having rules that do not erase any node, making it very easy to extend the grammar.

Exponential complexity of the parsers is a problem for graph grammars. In our grammar, all g_l are star graphs which makes subgraph matching easier. Currently, the system works in 1-10 seconds per expression for the majority of the expressions, though there are occasional times when the parser times out. We are in the process of implementing a best-first search algorithm where the rule generation is applied to the plausible interpretations (as indicated by the computed likelihoods) first.

ACKNOWLEDGMENT

This work is supported by The Scientific and Technical Research Council of Turkey with project number 107E271.

We thank the reviewers for their useful suggestions and criticism.

REFERENCES

[1] Buyukbayrak, H., Yanikoglu, B., and Ercil, A., "Online handwritten mathematical expression recognition," in [*Document Recognition and Retrieval Conference*], (2007).

[2] Ha, J., Haralick, R. M., and Phillips., I. T., "Understanding mathematical expressions from document images," in [*ICDAR*], (Aug. 14 1995).

[3] Garain, U. and Chaudhuri, B. B., "Recognition of online handwritten mathematical expressions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **34**(6), 2366–2376 (2004).

[4] Zanibbi, R., Blostein, D., and Cordy, J. R., "Recognizing mathematical expressions using tree transformation," *IEEE Trans. Pattern Anal. Mach. Intell* **24**(11), 1455–1467 (2002).

[5] LaViola Jr., J. J., *Mathematical Sketching: A New Approach to Creating and Exploring Dynamic Illustrations*, PhD thesis, Brown University, Department of Computer Science (2005).

[6] Chou, P. A., "Recognition of equations using a two-dimensional stochastic context-free grammar," in [*VCIP IV*], Pearlman, W. A., ed., *SPIE Proceedings Series* **1199**, 852–863 (1989).

[7] Pagallo, G. M., "Constrained attribute grammars for recognition of multi-dimensional objects," *LNCS* **1451** (1998).

[8] Chan, K. F. and Yeung, D. Y., "An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions," *Pattern Recognition* **33**, 375–384 (Mar. 2000).

[9] Eto, Y. and Suzuki, M., "Mathematical formula recognition using virtual link network," in [*ICDAR*], 762–767 (2001).

[10] Kosmala, A., Lavirotte, S., Pottier, L., and Rigoll, G., "On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars," in [*ICDAR*], (Sept. 1999).

[11] Lavirotte, S. and Pottier, L., "Optical formula recognition," in [*ICDAR*], (1997).

[12] Grbavec, A. and Blostein, D., "Mathematics recognition using graph rewriting," in [*ICDAR*], 417–421 (1995).

[13] Marzinkewitsch, R., "Operating computer algebra systems by handprinted input," in [*ISSAC*], 411–413 (1991).

[14] Bunke, H., "Graph grammars - a useful tool for pattern recognition?," in [*Graph-Grammars and Their Application to Computer Science*], *LNCS* **532**, 43–46 (1990).

[15] Fahmy, H. and Blostein, D., "A survey of graph grammars: theory and applications," in [*ICPR*], II:294–298 (1992).

[16] Celik, M., *Handwritten Mathematical Expression Recognition Using Graph Grammars*, Master's thesis, Sabanci University (2010).

[17] Vuong, B. Q., Hui, S. C., and He, Y. L., "Progressive structural analysis for dynamic recognition of on-line handwritten mathematical expressions," *Pattern Recognition Letters* **29**, 647–655 (Apr. 2008).

APPENDIX

$\mathbf{op} : +|-$
$\mathbf{eq} : = | < | > | \leq | \geq | \rightarrow$
$\alpha : (a...z)|\alpha|\beta|\gamma|\epsilon|\pi|\mu|\tau|\infty$
$\mathbf{n} : (0...9)$
$\mathbf{f} : sin|cos|tan|cot$
$\mathbf{g_1} : \sum | \prod$
$\mathbf{g_2} : \int$
$\mathbf{g_3} : lim$

$$\left.\begin{array}{r} \alpha^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>} \\ \\ \alpha_{<\alpha,n,n\alpha>} \end{array}\right\} \Rightarrow E_t$$

$$\alpha \ < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op} > \Rightarrow n\alpha$$

$$\mathbf{n}^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>} \Rightarrow E_t$$

$$\mathbf{g_1}^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>}_{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>} < \alpha, n, n\alpha, ..., E_{mt}, E_{op}, E_g > \Rightarrow \ E_g$$

$$\left.\begin{array}{r} \mathbf{g_2}^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>}_{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>} < \alpha, n, n\alpha, ..., E_g > \\ \\ \mathbf{g_2} < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op} > \end{array}\right\} \Rightarrow E_g$$

...