Mathematical formula recognition using graph grammar

Stéphane Lavirotte[a] and Loïc Pottier[a]

[a]INRIA Sophia Antipolis, 2004 Route des Lucioles, 06902 Sophia Antipolis, BP 93, France

ABSTRACT

This paper describes current results of Ofr (Optical Formula Recognition), a system for extracting and understanding mathematical expressions in documents. Such a tool could be really useful to be able to re-use knowledge in scientific books which are not available in electronic form. We currently also study use of this system for direct input of formulas with a graphical tablet for computer algebra system softwares. Existing solutions for mathematical recognition have problems to analyze two dimensional expressions like vectors and matrices... This is because they often try to use extended classical grammar to analyze formulas, relatively to baseline. But a lot of mathematical notations do not respect rules for such a parsing and that is the reason why they fail to extend text parsing technic. We investigate graph grammar and graph rewriting as a solution to recognize two dimensional mathematical notations. Graph grammar provide a powerful formalism to describe structural manipulations of multi-dimensional data. The main two problems to solve are ambiguities between rules of grammar ([1] for theorems) and construction of graph.

Keywords: Formula recognition, optical character recognition, document understanding, document recognition, graph grammar, graph rewriting

INTRODUCTION

For many years, transmission and storage of information have been by paper documents. Since the emergence of computer science and binary information treatment, documents are originated on computers, but it's not clear if it decreased or increased the use of paper. However, documents are still printed for mass edition (such as for books), sending mailing…and, in many case, you can not have electronic sources of documents you received. The ultimate solution would be for computers to deal with these paper documents and turn them into an electronic form. So that, in the late 1980s, fast computers, large computer memory, and inexpensive dedicated hardware increased the interest in document image processing and analysis. A lot of researches were made before and are still done on pattern recognition, document analysis…Currently good results are expected for text or form recognition, for document analysis, but some components of documents are not treated at all to extract usable information. This is the case of formula for example. Some new technic allow to separate and identify different components of a document.[2] No commercial OCR deal with such expressions ; mathematical formula are sometimes clearly located but never analyzed or recognized.

Parsing 2D expressions is much more difficult than parsing strings because mathematical expressions, or other 2D languages, differ greatly from text. A line of text is one-dimensional and discrete: characters are placed one after another on the same line, the only problem to deal with being line breaking. But symbols in mathematical expressions may be *under*, *upper on the right and far*, *included in another*, etc, with continuous distances. Many works have been done since the sixties on parsing two dimensional expressions and the development of methods for recognizing mathematical expressions has been a subject of growing interest in last years[3] ,[4] ,[5] .

There is a wealth of mathematical knowledge in books that can be potentially very useful in many computational applications. But these material are not available in a computer usable form. Currently, the only way to use mathematical informations in a document is to re-type formulas on keyboard to be able to add it in Computer Algebra System (CAS) or in any application using mathematical input. To incorporate such informations into systems (CAS or formula database), there is too much typing work. If one also wants to bypass the relatively unfriendly input interfaces of most CAS, one can dream of writing formulas with a pen on the screen or on a board. In these cases, the problem to solve is : How to build the syntax tree of a formula just with graphical informations (recognized characters and their position) ?

Other author information (correspondence):
S.L.: E-mail: Stéphane.Lavirotte@sophia.inria.fr; WWW:http://www.inria.fr/safir/slavirot/
L.P.: E-mail: Loic.Pottïer@sophia.inria.fr; WWW:http://www.inria.fr/croap/personnel/Loic.Pottier/home.html

After this introduction, we will present :

First we introduce our method, design of our prototype and discuss other approaches of other researches in the domain.

In a second part, we briefly present the preprocessing step to analyze image, to extract character and needed features.

In the third section, we present the methodology we have used for syntax analysis. After a short introduce of notion of *graph representing a formula*, we explain the geometric construction of graph from the result of the OCR step. Finally we present the formalism of our graph grammar, properties allowing to eliminate ambiguities, and show how to use them to parse formulas.

The fourth part describes briefly the implementation, shows some examples which are currently well parsed by our prototype and then, we conclude in describing future works.

1. DESCRIPTION OF OFR DESIGN

We describe the design and first implementation of *Ofr* (*Optical Formula Recognition*), a system for extracting and recognizing mathematical expressions in printed documents. The system described in this paper is based on three different components, each one giving just the necessary information to the next process. Here we suggest a scheme to really separate the different steps of building the syntax tree from the graphical informations. Figure 1 represents the *Ofr* organization.

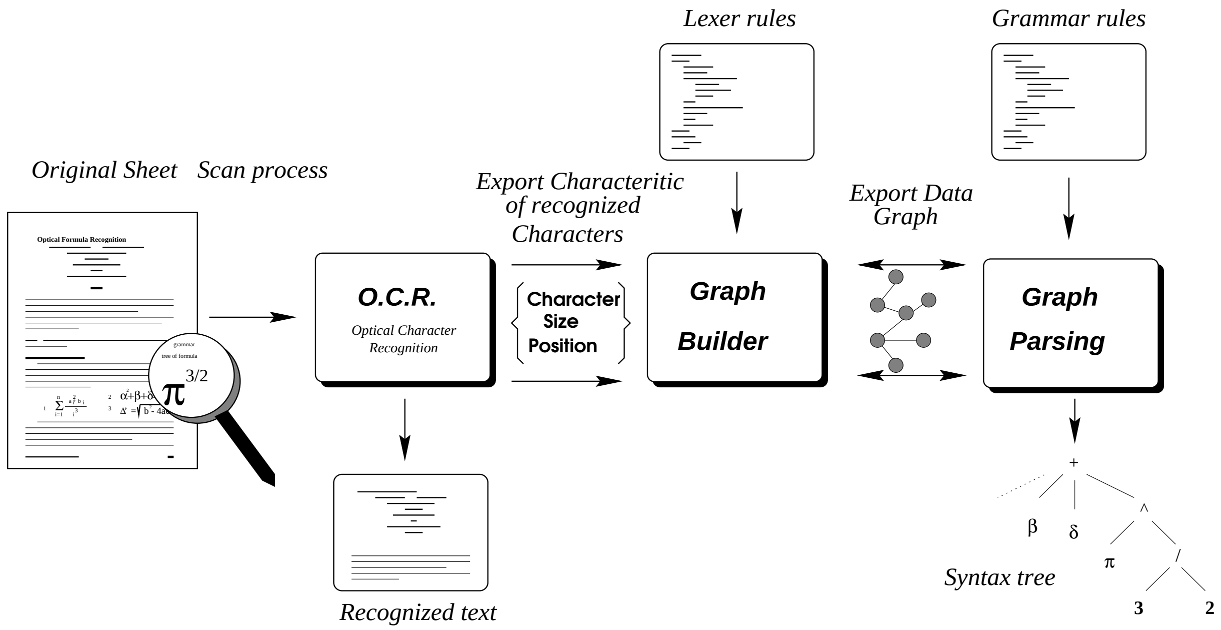

Figure 1. The *Ofr* architecture

So, our approach clearly separate character recognition, geometrical treatment, and grammatical treatment. This approach was used at the beginning of OCR study, but current commercial program prefer to use syntax or semantic rules during the recognition process. We chose this approach because it's a simple one and for the moment, we don't study problem of miss recognition and errors correction. This separation between all the components allow us to be able to swap used OCR and allow a large set of tests on generated images (by LaTeX for example), on real scanned images, and also with handwritten recognition software. We do not have results yet on this third category of test but we make some tests of usability.

From the result given by OCR (*Optical Character Recognition*) step applied to a bitmap representation of a scanned formula, we build a graph encoding relative positions of characters (geometrical treatment), and then use a context dependent graph grammar to reduce it to an irreducible form. Then nodes contain abstract syntax trees of recognized formulas. We introduced a method to automatically complete a given context free graph grammar by adding contexts to rules, in order to remove ambiguities. This method is based on a "critical pairs" approach in the sense of Knuth-Bendix algorithm.

2. OCR

The OCR (*Optical Character Recognition*) step is very important, and researches on this subject have lead to good solutions, at least for printed characters and also for handwritten ones with HMM's method. In this paper, we won't discuss about character recognition problems. We have completely disconnected this problem from the rest of our study.

We just assume that there is an OCR process which is able to separate formulas from the rest of the document. This operation is not very hard to do because the density of characters in formulas are not the same as text. An other criterion is the two dimensional structure of mathematical formulas. Horizontal and vertical projections of the page would give interesting information too. Current researches have lead to usable results.

Then for each isolated area of formula the process should give output informations about symbols present on the sheet. For each character, we expect to have :

- recognized symbol.
- coordinates of the bounding box of each characters (the position of the symbol on the sheet in absolute or relative coordinates).
- font size of the character. For this point we don't suppose that the OCR is able to compute the exact size in points of the font, but a relative size between all the characters.
- reference point of the character (typically the baseline of the character). This information can be approximately generated if the used OCR is not able to give it.

We use a simple OCR developed at our laboratory. This software is able to learn some given fonts (for example the LaTeX ones) and after to recognize formula symbols and characters printed with one of the learned font. A font is learned by giving an image of characters we want the OCR to learn and the corresponding ASCII description for each one. So that, it returns all the needed informations to the subprocess. For example, for the following simple formula $(a^2 + b)$, it returns as output:

Symbol	Bounding Box	Baseline	Size
2	1246,454,1258,472	1246,472	7
b	1316,461,1330,490	1316,490	10
+	1275,466,1302,492	1275,489	10
a	1224,471,1243,490	1224,490	10

All these informations will be used to construct a data graph as we will see in next section.

3. SYNTAX ANALYSIS

3.1. Definition of graphs

We introduce an intermediate combinatorial structure, between the recognized symbols with their positions in the plane, and the tree of formula. It is a graph, nodes being characters, and edges being relative positions of these graphical objects in the paper sheet. Then, using a graph grammar, the parsing algorithm will update and reduce the graph of data into a single node which will contain the syntax tree of the formula.

Of course, the construction of the graph is difficult, but we think that the separation of geometric and syntax is very important to understand and solve the problem of parsing 2D expressions, like mathematical ones.

Let us precise now our notion of graph.

Every object will be represented by terms or finite sets of terms. As usual, the set $T(F, V)$ of terms is inductively defined by a set F of functional symbols of fixed arity and a set V of variables: variables are terms, and if $t_1, \ldots, t_n$ are terms, and f is a n-ary functional symbol of F, then $f(t_1, \ldots, t_n)$ is a term. We note $Var(t)$ the set of variables occurring in a term (or a finite set of terms) t.

We use uppercase symbols for functional symbols and lowercase ones for variables.

Vertices, edges and graphs are represented as follows :

- a **vertex** is a term $V(t, v, i)$ where:
  - t is its lexical type, e.g. "Operator", "Variable", "Digit",etc.
  - v is its value, typically a mathematical expression in term form, e.g. x, $Plus(x, (Mult(2, y)))$, etc.
  - i is an identifier, distinguishing distinct occurrences of the same mathematical expression.
- an **edge** is a term $E(t, v_1, v_2)$ where:
  - v_1 and v_2 are vertices.
  - t is a type of edge, i.e. a term $L(d, w)$, d being a graphical directions (e.g. "Left", "Right", "Top", etc), and w being a weight, encoding the relative proximity of two symbols in the plane.
- a **graph** is a finite set of edges:

  $\{E(t_1, v_{11}, v_{2,1}), \ldots, E(t_n, v_{1n}, v_{2,n})\}$. The set $\{v_{ij}\}$ being the set of vertices of the graph. For simplicity, we suppose that graphs are connected and have at least one edge *.

The next section will detail how the data graph is constructed and the interaction between *graph builder* and *graph grammar* parser.

3.2. Construction of data graphs

Extending the usual methodology of string languages analysis, we use the notion of lexical units, or token. In our case, a token is basically a symbol of the sheet, and will be more complicated expression during the parsing process.

The *graph builder* constructs a graph with all tokens. Oriented links between vertices are deduced from graphical informations. In fact, this step is a generalization of the only two links *before* and *after* determining relative character position in a computer input string.

Theses graphic oriented links are intended to capture all useful geometric informations of character relative positions. The main problem in graph building is to find a good trade-off between two extreme cases:

- a graph with too many links will represent more than one formula, and then lead to inconsistency.
- a graph with few links will not contain sufficient informations to build the formula.

The third possible problem is not to build false links between edges. For example if the algorithm build a right link instead of a upper right one, the analyzed result won't be correct. So graphical algorithms to determinate position of object should be well written. Here is an overview of the general algorithm to build the data graph.

For each symbol of the sheet, we try to link it in all the 8 directions of the plane (*left* (**l**), *right* (**r**), *top* (**t**), *bottom* (**b**), *top-left* (**tl**), *top-right* (**tr**), *bottom-left* (**bl**), *bottom-right* (**br**)) with the closest symbols. A last type of connections between 2 symbols should not be forgotten : *in* (**i**) which is used for the square root for example.

This give us a set of possibles connections between symbol and neighbors in all directions. Then, before creating these links between symbols (i.e. edges between the corresponding vertices), we apply some test to know is the edge we want to create as a sense.

First, we use a criterion based on type of symbol. In the lexer we associate type to particular symbol and for each type, we can specify what sort of links are not allowed. For example, here is a very simple lexer :

*Then every vertices appears in at least one edge. This is not a restriction : we can add a generic vertex, connected to every vertex with generic edges.

		List of forbidden	
Reg-expr	Type	outgoing edges	incoming edges
[0-9]	Digit	()	()
"Sigma"	Sum	'(tl bl tr br)	'(tr br)
...			

The second criterion is based on the manner to read or write mathematical expressions. For text, we write from left to right without line changing excepted at the end for line breaking. Mathematical expressions are more difficult because have 2D placement. But a lot of used notations are based on normal reading (from left to right). So symbols which are on top-left or bottom-right or... are very close from the reference symbol. In the other hand, horizontal links between symbols can be far-off. On this establishment of fact, we have introduced a notion of gravity between symbols. So, for all neighbors of a symbol, we calculate the force of magnet between the 2 symbols. Gravitational force should be very high for upper, lower or diagonal links but can be less for horizontal ones. Figure 2 shows a graphical representation of an isopentional of the considered box.

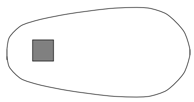

Figure 2. Representation of gravitational isopotential for a symbol box

We introduced a grid like structure to be able to have a good algorithm complexity so that it's possible to define neighbors of a symbol not with a $O(n^2)$ complexity but in a constant time (in fact depending on the density of symbols near the considered one).

The next step is now to define a type of grammar and a parsing method to use this combinatorial structure in order to derive tree (i.e. term) representations of formulas.

3.3. Graph grammar

Graph grammar provide a useful formalism to describe structural manipulations of multi-dimensional data. They were introduced in[6] to solve picture processing problems, and are studied in a theoretic point of view (e.g. in[7] ,[8]), or in a more practical one ([9] ,[10]). To have a good overview of this subject, see[11] ,[12] .

Graph grammars are used to parse and generate graphs. A graph grammar is specified by a start graph (the one we have constructed during the previous step) and a set of production rules. The role of the rule is to replace the matched subgraph by another one. This process depends on a specification on the desired embedding, this means that there is different ways to replace the matched subgraph.

We use context-sensitive graph grammars. A rule of grammar expresses that a sub-graph of the graph can be collapsed into a new vertex (representing the sub-formula) if some conditions are verified by the involved tokens. Terminal symbols represent symbols detected on the sheet by the OCR program and non terminal symbols represent a recognized expression.

3.3.1. Definition

The precise forms of rules and grammars are the following:

- a **rule** is a term $V \leftarrow G, C$ where:
  - V is a vertex, called the "production" of the rule.
  - G is a graph, called the "pattern" of the rule.
  - C is a finite set of graphs, called the "context" of the rule. We will precise this point in the next section.
- a **grammar** is a finite set of rules.

Given a graph representing a formula (its vertices are symbols and edges are graphical links between them), rules are intended to rewrite it by replacing sub-graphs by vertices whose values are term forms of the recognized sub-formulas. This process uses matching and replacement in a way that we precise below.

First, we recall the notions of substitution and term matching :

- a **substitution** is an endomorphism of $T(F, V)$, i.e. an application σ verifying $\sigma f(t_1, \ldots, t_n) = f(\sigma t_1, \ldots, \sigma t_n)$ for all f in F and all terms $t_1, \ldots, t_n$. A substitution σ is uniquely determined by its restriction $\sigma_{|V}$ to the set of variables.
- a term t **matches** a term t', noted $t \leq t'$ iff there exists a substitution σ such that $\sigma t = t'$.

Matching of finite sets of terms is defined by :

$$\{t_1, \ldots, t_n\} \leq \{t'_1, \ldots, t'_m\} \Leftrightarrow \exists \sigma \ \{\sigma t_1, \ldots, \sigma t_n\} = \{t'_1, \ldots, t'_m\}$$

A rule $r = V \leftarrow G, C$ **rewrites** a graph G_1 into a graph G_2, noted $G_1 \rightarrow_r G_2$ iff there exists a substitution σ, a sub-graph G' of G_1 (i.e. $G' \subset G_1$), such that:

- $\sigma G = G'$.
- for all graph H in the context C, there is no substitution τ such that $\tau_{|Var(G)} = \sigma_{|Var(G)}$ and $\tau H \subset G_1$.
- G_2 is obtained by collapsing G' into σV, i.e. removing in G_1 all edges of G' and replacing in G_1 all the vertices of G' by the vertex σV.

3.3.2. Contexts of rules

One of the main problem with grammar and rewrite rules is the existence of ambiguities: two rules can rewrite an object into two distinct objects. Suppression of ambiguities can be made for example by using priorities, case analysis on pattern of rules, or by Knuth-Bendix completion. These techniques hardly apply to our case, this is why we use **contexts** in rules: given a graph grammar which leads to ambiguities, our goal is to add contexts to its rules to remove these ambiguities, as automatically as possible.

When two rules can apply to two sub-graphs of graph which have disjoint sets of vertices, there is no ambiguity: applications of the to rules commute.

Ambiguities can appear when the two patterns of the rules can be superposed:

- two graphs G_1 and G_2 can be superposed iff there exist σ_1 and σ_2 such that $\sigma_1 G_1$ and $\sigma_2 G_2$ have a common vertex. We note $S(G_1, G_2)$ the set of couples of such substitutions, called **superpositions** of G_1 and G_2.
- given two rules $r_i = V_i \leftarrow G_i, C_i$, $i = 1, 2$, the set $A(r_1, r_2)$ of ambiguities of r_1 and r_2 is defined as the subset of $S(G_1, G_2)$ formed by couples (σ_1, σ_2) such that the two rules can apply to the graph $\sigma_1 G_1 \cup \sigma_2 G_2$, i.e. $\forall i = 1, 2$, $\forall H \in C_i$, there is no substitution τ such that $\tau_{|Var(G_i)} = \sigma_{i|Var(G_i)}$ and $\tau H \subset \sigma_1 G_1 \cup \sigma_2 G_2$.

The set $S(G_1, G_2)$ can be infinite, but "minimal" superpositions are in finite number, as shown by the next propositions.

In this paper we omit proofs and description of construction of contexts of rules, which are more technical than difficult. In all cases, all can be found in[1] .

3.4. Parser

The parsing algorithm we use is a bottom-up algorithm. Two main reasons motivated this choice. Firstly, a sheet can contains many formulas, and a bottom-up approach allow local treatment of the sheet. This also correspond to a human perception of formula, even if we often sue a global view of a formula before reading its components. Trying to simulate this global view by top-down parsing is possible, but we think that this approach will lead to a combinatorial explosion and an exponential parsing which is generally the case in two dimensional parsing algorithms in literature. Secondly, geometric parameters of a formula deduced from those of its components are given by functions which are very difficult to inverse. In particular, the bounding box can not be split into sub-rectangles as we can easily make a partition from a string into sub-strings in a top-down parsing. Moreover, to be able to make top-down parsing suppose to determinate graphic links between sub-formulas and again determine geometric parameters of components from parameters of formulas.

The parsing algorithm consists in applying the first rule which can be applied in regards to the contexts. All the matched nodes are collapsed as specified in the rule. Then, we update all the outgoing links of matched sub-graph with the neighbors.

A straightforward analysis shows that, if n is the number of symbols on the sheet, then the time complexity of the parsing algorithm is at most $O(n \log n)$ with the grid structure presented in graph construction section.

4. CURRENT RESULTS

4.1. Implementation

The current developed software to recognize mathematical expressions runs under UNIX system. The supported systems are Linux machines (for example PC), Sparc computers running SunOS 4.x and Dec UNIX.

We use a Hewlett Packard ScanJet 4c scanner to scan the mathematical expressions document and save it as a binary image file. The software used to drive the scanner is xvscan, an extension to the famous xv software.

OCR used currently is a small package written in Java by our team. This software is able to learn font sets and after to recognize characters printed with these fonts. This package is about 7500 lines of Java.

The Graph Builder and Graph Grammar package are currently implemented in KLONE, a Common Lisp dialect. KLONE has been designed specially to be embedded in C applications. Calling a C function or accessing a C structure from KLONE interpreter is straightforward. KLONE is among the most efficient embeddable Lisp interpreter and still a very small process (about 200Kb on a Sparc Station running SunOS 4.x). All these advantages were useful to quickly develop these experimental packages on graph grammar. Graph Builder and Graph Grammar are about 9000 lines of KLONE.

4.2. Samples

In this section, we present some samples which are correctly analyzed by our *Ofr* prototype. For history, the first real sample which as been analyzed is the following :

$$\frac{\frac{x^{-201}+y^{523}}{abce}}{(x^2+y^2)(x^3+y^3)}$$

We won't detail grammar and problems on this sample but on the next which is a bit more difficult.

$$\left(\frac{1}{x^2+1}\right)^{(n)} = (-1)^n.n!\frac{\sum_{0\leq 2p\leq n}(-1)^p C_{n+1}^{2p+1} X^{n-2p}}{(X^2+1)^{n+1}}$$

The lexer needed is not very important. This part depends on the output of the used OCR. For example if the OCR return a special code for some characters, we must write a special rule to give to this code the good type. The used OCR return sigma name for the sigma symbol. So, the lexer should contain the following declaration :

```
"Sigma"        Sum     '(tl bl tr br)      '(tr br)
```

The lexer part permit to adapt the graph builder to the used OCR.

The grammar to analyze this sample contains 15 rules. There is different kinds of operators declared in the grammar :

- **linear**:
  - *postfix* for factorial (!)
  - *infix* for addition subtraction... $(+, -, *\dots)$
  - *prefix* for negation (unary minus).
- **vertical**: Like divide operator (/).
- **implicit**: Like products, power, indices ($2x$, x^y, x_y).
- **2D**: Sum ($\sum$) and binomial (C_n^p).

Critical cases are for the sum argument. There is some small characters with several changes of baseline for indexes, power. Adding to all this, there is implicit multiplication between the 3 terms. This part is not hackneyed to correctly analyze. This can be correctly understood because of the contexts of grammar rules. It starts to collapse the implicit multiplication between 2 and p and then collapse all the add, minus, unary minus operators. Then it collapses C_n^p, power and to finish it do the implicit multiplication between all the 3 elements.

To have an overview of how the algorithm works on this sample, you can have a look to :

http://www.inria.fr/safir/slavirot/Ofr/ . You would be able to see an animation of the parsing of this formula.

5. CONCLUSION AND FURTHER WORKS

We have presented in this paper a method and a system to recognize scanned mathematical formulas. Originalities of our approach are: on a theoretical level, this allows us to precisely define a method to remove ambiguities of graph grammars. On a practical level, we have a first implementation of the method, which works on various complex formulas printed in LaTeX for example, obtained from bitmap images of formulas, with good time complexity. Defined grammar for these formulas are not trivial, using more than 50 operators, with many kinds of constructions:

- linear operators: prefix, postfix, infix, parenthesis.
- vertical operators: fractions, etc.
- 2D tree-operators: indexed sums, integrals, limits.
- implicit operators: product, power, indices, arrays.

For most grammars dealing with these constructions, we are able to remove correctly ambiguities, with the presented criterion. In some cases, we need heuristics, but our aim is to mechanize this process.

In real applications, OCR is not able to recognize each characters with 100% of success. We will study how to incorporate errors detection and correction in our graph grammar as introduced in[13] . We will test also the possibility of adapting our approach to handwritten character recognition to be able to write formulas on a tablet and to use this method to bypass keyboard typing of formulas for application needed mathematical input.

REFERENCES

1. S. Lavirotte and L. Pottier, "Optical formula recognition," in *Fourth International Conference on Document Analysis and Recognition*, I. C. Society, ed., *Proc. SPIE* **1**, pp. 357–361, 1997.
2. A. K. Jain and B. Yu, "Page segmentation using document model," in *Fourth International Conference on Document Analysis and Recognition*, I. C. Society, ed., *Proc. SPIE* **1**, pp. 34–38, IEEE Computer Society, 1997.
3. R. J. Fateman, T. Tokuyasu, B. P. Berman, and N. Mitchell, "Optical character recognition and parsing of typeset of mathematics," in *International Symposium on Symbol and Algebraic Computation*, *Proc. SPIE* , July 1994.
4. H. M. Twaakyondo and M. Okamoto, "Structure analysis and recognition of mathematical expressions," in *Third International Conference on Document Analysis and Recognition*, (Montréal), Aug. 1995.
5. A. Grbavec and D. Blostein, "Mathematics recognition using graph rewriting," in *Third International Conference on Document Analysis and Recognition*, vol. 1, pp. 417–421, (Montréal), Aug. 1995.
6. J. Pfaltz and A. Rosenfeld, "Web grammars," in *Proc. First international Joint Conference on Artificial Intell igence*, pp. 609–619, (Washington), 1969.
7. B. Courcelle, "An axiomatic definition of context-free rewriting and its application to nlc graph grammars," tech. rep., Université de Bordeaux, Feb. 1987.
8. J.-C. Raoult and F. Voisin, "Set-theoretic graph rewriting," tech. rep., IRISA, Apr. 1992.
9. H. Bunke, "Graph grammars as a generative tool in image understanding," in *Graph Grammars and their Application to Computer Science*, H. Ehrig, M. Nagl, and G. Rozenberg, eds., vol. 153 of *Lecture Notes in Computer Sciences*, pp. 8–19, Springer-Verlag, Oct. 1982.
10. H. Fahmy and D. Blostein, "A graph grammar for high-level recognition of music notation," in *Proceedings of First International Conference on Document Analysis and Recognition*, vol. 1, pp. 70–78, IEEE Computer Society, (Saint Malo, France), Oct. 1991.
11. H. Fahmy and D. Blostein, "A survey of graph grammars: Theory and applications," in *Proceedings of the 11th International Conference on Pattern Recognition*, vol. The Hague, Netherlands, IEEE Computer Society, Sept. 1992.
12. M. Nagl, "A tutorial and bibliographical survey on graph grammars," in *Workshop on Graph-Grammars and their Application to Computer Sc ience and Biology*, vol. Lecture Notes in Computer Science, pp. 70–126, (Bad Honnef), Nov. 1978.
13. Y. A. Dimitriadis, J. L. Coronado, and C. de la Maza, "A new interactive mathematical editor, using on-line handwritten symbol recognition, and error detection-correction with an attribute grammar," in *Proceedings of First International Conference on Document Analysis and Recognition*, vol. 1, pp. 885–893, IEEE Computer Society, (Saint Malo, France), Oct. 1991.