Optical Formula Recognition

Stéphane Lavirotte *

Safir Team
INRIA Sophia Antipolis
Sophia-Antipolis, FRANCE 06560

Loïc Pottier

Safir Team
INRIA Sophia Antipolis
Sophia-Antipolis, FRANCE 06560

Abstract

This paper describes the design and the first steps of implementation of Ofr (Optical Formula Recognition), a system for extracting and understanding mathematical expressions in printed documents. Our approach clearly separate OCR step, geometrical treatments and syntactic analysis. In this paper we focus on the third part: we define a class of context-sensitive graph grammars for mathematical formulas, study their properties and show how to remove their ambiguities (by adding contexts in rules) to define efficient parsing. This method is based on a "critical pairs" approach in the sense of Knuth-Bendix algorithm.

Introduction

The paper is organized as follows : first we introduce the domain of mathematical formulas recognition, and discuss works of the literature. In second part, we briefly present our objectives and the motivations of our researches. In third section, we present the methodology we have used. The fourth part present the formalism of our graph grammars, properties allowing to eliminate ambiguities, and show how to use them to parse formulas. The fifth part describes briefly the implementation. Then, we conclude in describing future works.

1 Mathematical formulas recognition

There is a wealth of mathematical knowledge that can be potentially very useful in many computational applications. But this material is not available in electronic form. All this knowledge is in mechanical, physical and mathematical books which are references in the domain from many years. Other newer sources are publications, articles, filled of useful informations which are often difficult to get sources. Actually, the only way to use this mathematical informations 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. For automatization of this job, the problem to solve is : *How to build the syntax tree of a formula just with graphical informations (characters and their positions) ?*

Many works have been done since the sixties on parsing two dimensional expressions. Parsing 2D expressions is 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. But symbols in mathematical expressions may be *under*, *upper on the right and far*, *included in another*, etc, with continuous distances.

Different methods were used to solve this problem with various success ([1], [3], [6], [8], [10], [13],[14]) which can roughly be classified in: syntax directed recognition ("local" context-free grammar approach and heuristics) and geometry directed recognition using layout structure of symbols.

This methods have problems to manage "non-linear" formulas like matrices or systems of equations. Also, parsing techniques in literature have often exponential complexity, when they use 2D grammars derived from classical context free string grammars, with geometrical predicates. Exploration and backtracking, used in these parsers, lead to exponential complexity.

2 Objectives

Our aim is to start from scanned images (*"bitmaps"*) of documents containing formulas and to extract, read and parse them to be able to re-use them in other applications. As we mentioned in introduction, it would be very interesting to have such a tool. Different fields of applications can be considered : build a base of knowledge (Database of mathematical formulas or CAS database), copy/paste between a viewer of Postscript and a CAS or an editor, complete actual professional OCR which do not have formula recognition capabilities.

*also I3S, CNRS URA 1376, Université de Nice Sophia Antipolis

3 Design of *Ofr*

The *Ofr* system described in this paper is based on three different components, each one giving just the necessary information to the next process.

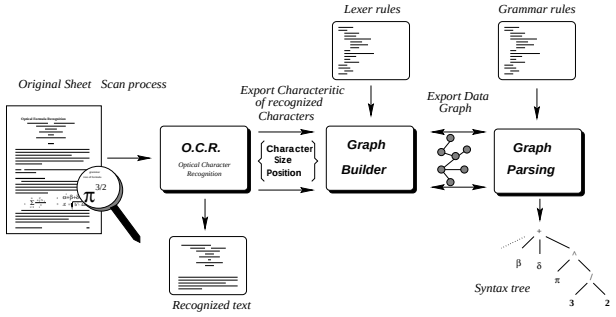

Figure 1: The *Ofr* architecture

3.1 OCR (*Optical Character Recognition*)

The OCR step is important, and researches on this subject have lead to good solutions, at least for printed characters. In this paper, we won't discus about the recognition problem. We assume that their is a process which gives as output symbols of the sheet and informations about them. We need the bounding box of each symbol (position in absolute or relative coordinates), size of the character. For size, we don't suppose that OCR can give the absolute size in points, but a relative size between all characters. Eventually reference point of the character could be helpful.

3.2 Definition of graphs

We introduce an intermediate combinatorial structure, between recognized symbols with their positions in the plane, and the tree of formula. It is a graph, linking graphical objects of the paper sheet. This graph contains geometrical informations about all the elements of formula and their relative positions.

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. Theses graphic oriented links are intended to capture all useful geometric informations of character relative positions. In fact, this step is a generalization of the only two links *before* and *after* determining relative character position in a computer input string. 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.

Every object will be represented by terms or finite sets of terms. The set $T(F, V)$ of terms is inductively defined by the set F of functional symbols of fixed arity and the set V of variables: variables are terms, and if $t_1, .., t_n$ are terms, and f is a n-ary functional symbol of F, then $f(t_1, .., t_n)$ 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.

- a **vertex** is a term $V(t, v, i)$ where: t is lexical type ("Letter", "Digit", etc), v is value (typically a mathematical expression in term form : x, $Mult(2, y)$, etc), i is an identifier.
- an **edge** is a term $E(t, v_1, v_2)$ where: v_1 and v_2 are vertices, t is type of edge, i.e. a term $L(d, w)$, d being a graphical directions ("Left", "Top", etc), and w being a weight, encoding relative proximity of two symbols.
- a **graph** is a finite set of edges : $\{E(t_1, v_{11}, v_{2,1}), .., E(t_n, v_{1n}, v_{2,n})\}$. 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 [1].

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 (term) representations of formulas.

4 Structure analysis : graph grammar

Graph grammars provide a useful formalism to describe structural manipulations of multi-dimensional data. They were introduced in [11] to solve picture processing problems, and are studied in a theoretic point of view [12], or in a more practical one [2], [4]. To have a good overview of this subject, see [5], [9].

A graph grammar is specified by a set of production rules. The role of rules is to replace matched subgraph by another one. This process depends on a specification on the desired embedding, this means that there is different ways to replace matched subgraph.

4.1 Definition

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 subformula) if some conditions are verified by the involved tokens. Rules and grammars are defined as following:

[1] 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.

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

Given a graph representing a formula, 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 $\sigma f(t_1, .., t_n) = f(\sigma t_1, .., \sigma t_n) \forall f$ in F and all terms $t_1, .., t_n$. 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 :

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

Then 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 substitutionτ such that $\tau_{|Var(G)} = \sigma_{|Var(G)}$ and $\tau H \subset G_1$; and 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.

4.2 Contexts of rules

One of the main problems 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 which have disjoint sets of vertices, there is no ambiguity: applications of the two 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 next propositions. First we define a pre-order on couples of substitutions :

Definition 1

$$(\sigma_1, \sigma_2) \leq (\sigma'_1, \sigma'_2) \Leftrightarrow \exists \tau, \sigma'_1 = \tau \circ \sigma_1, \ \ \sigma'_1 = \tau \circ \sigma_1$$

Proposition 1 *Given two graphs G_1 and G_2, there exists a finite subset S_0 of $S(G_1, G_2)$ such that:*

$$\forall c \in S(G_1, G_2), \ \exists c' \in S, c' \ll c.$$

We omit proofs by lack of space (see [7] for details).

Proposition 2 *There exists an unique (up to renaming of variables), minimal (for $\leq$) and finite set of superpositions of two graphs.*

We will note $S_0(G_1, G_2)$ this minimal set.

Because of contexts, the set of ambiguities of two rules has a more complicated structure than the set of superpositions of their patterns. But the following property will help us to describe it:

Proposition 3 *Let $r = V \leftarrow G, C$ be a rule, $r' = V \leftarrow G, \emptyset$ be the same rule with empty context, G_1, G_2 two graphs, such that $G_1 \rightarrow_{r'} G_2$ and $G_1 \not\rightarrow_r G_2$.*

Let ρ be a substitution.

Then $\rho G_1 \rightarrow_{r'} \rho G_2$ and $\rho G_1 \not\rightarrow_r \rho G_2$.

Let us given two rules r_1 and r_2 with patterns G_1 and G_2, and a superposition (σ_1, σ_2) of $S(G_1, G_2)$. If we remove contexts of the rules, then both apply to the graph $\sigma_1 G_1 \cup \sigma_2 G_2$. Suppose now that (σ_1, σ_2) is not an ambiguity of r_1 and r_2. Then one of the two rules (with their contexts) does not apply to $\sigma_1 G_1 \cup \sigma_2 G_2$. The last proposition implies that for every substitution τ, the superposition $(\tau \circ \sigma_1, \tau \circ \sigma_2)$, is not an ambiguity of r_1 and r_2. More briefly, we have then:

Proposition 4 *If $c \in S(G_1, G_2) \setminus A(r_1, r_2)$ and $c \leq c'$ then $c' \in S(G_1, G_2) \setminus A(r_1, r_2)$.*

This means that the complement of $A(r_1, r_2)$ in $S(G_1, G_2)$ has the same nice "cone" structure as $S(G_1, G_2)$. In particular, we have:

Corollary 1 *If the minimal superpositions of the patterns of two rules are not ambiguities of the rules, then the rules have no ambiguities.*

We will exploit now this property to define an method that, given a grammar with ambiguities, adds contexts to its rules in order to obtain a new grammar without ambiguities.

4.3 Construction of Contexts

We can now give the general formulation of the method which removes ambiguities of context-free graph grammar by adding contexts.

Let $\mathcal{G} = \{r_1, .., r_n\}$ be a graph grammar, with $r_i = V_i \leftarrow G_i, \emptyset$.

Let $(\sigma_{k1}^{ij}, \sigma_{k2}^{ij}), k = 1..a_{ij}$ be all the minimal superpositions of G_i and G_j. Because contexts are empty, these superpositions coincide with ambiguities.

Suppose that for each minimal superposition (when two rules can be applied simultaneously) we have a method to choose the good rule to apply. It is equivalent to give a function C such that $C(i, j, k) \in \{1, 2\}$: if $C(i, j, k) = 1$, this means that we want to prevent application of rule r_j in the ambiguity $(\sigma_{k1}^{ij}, \sigma_{k2}^{ij})$. We achieve this goal by adding context $\sigma_{k1}^{ij} G_1$ to rule r_j.

Doing this for all rules and all minimal superpositions, we define the new grammar $\mathcal{G}' = \{r'_1, .., r'_n\}$, where:

$$r'_i = V_i \leftarrow G_i, \bigcup_{j=1...n} \{\sigma_{k2}^{ij} G_j | k = 1 \dots a_{ij},\ C(i, j, k) = 2\}$$

By Corollary 1, we have then:

Proposition 5 *The grammar $\mathcal{G}'$ has no ambiguity.*

Figure 2 show a representation of a context-free grammar $\mathcal{G}$ which has two rules, one for addition $z + t$ and one for implicit power x^y.

Figure 2: Graphs for rules r_+ and $r_\wedge$

For rule r_+ : the first part is the description of "production" of rule is the node in which the matched graph will be rewrite. Second part is graph to match ("pattern"). This graph is constituted of two edges, with their types (direction and weight) and nodes.

Suppose we have the formula $A^2 + B$, given by the graph represented by figure 3

There is an ambiguity because we can apply the two rules to the data graph. If we apply r_+, vertices A, $+$ and B collapse, and we obtain a graph representing

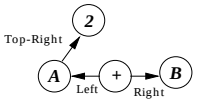

Figure 3: Graph for $A^2 + B$ formula

the formula $(A + B)^2$. If we apply $r_\wedge$, then we obtain a graph representing the formula $(A^2) + B$. The right choice is clearly to apply $r_\wedge$: we have to prevent the application of r_+ in this case.

Here are, in graphic representation, the four minimal superpositions of rule r_+ and $r_\wedge$ (data graph is a particular case of the first superposition) :

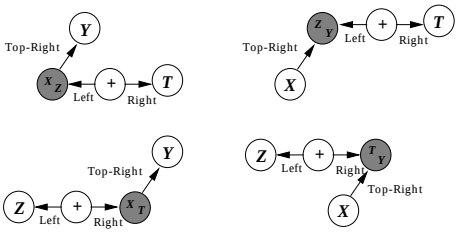

Figure 4: All the overlaps for rules r_+ and $r_\wedge$

We have defined some general criterions to obtain the C function. Firstly the mathematical priority of the described operator. Secondly a graphical information. In our example, $\wedge$ as a greater priority than $+$, but this doesn't mean that this rule should always be applied before the other one. This priority is right for linear case. In case of $\wedge$ operator, there is implicit parenthesis on arguments and this is true for all such operators with arguments on different level like $x^{(y+z)}$, $\sum_{(i=1)}^{(n^2)} x_i$, $\frac{(x+y)}{(z+t)}$, ... With these two simple criterions we are able to define which rule should be applied in each case of ambiguity.

In our example, for the first superposition, we do not want to apply r_+ first, then we just add the pattern of rule $r_\wedge$ to contexts of r_+. So that, r_+ won't be applied if there is a power on its argument.

Doing this for all superpositions, we obtain a context-dependent grammar $\mathcal{G}'$. This grammar works well on formulas like $a^2 + b$, $x^{a+b^2} + y^{c^{i+j^2}+d}$, etc.

4.4 Parser

The parsing algorithm we use is a bottom-up algorithm. 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.

5 Implementation

We use a HP ScanJet 4c scanner to scan mathematical expressions document and save it as a binary image file. OCR used actually is a small package (about 1500 lines of C) written in our team which is able to recognize printed documents by LaTeX. One of our aim is to replace this by a more generic OCR, which would be able to learn from a file and it's ascii translation, then recognize a document in the same fonts.

The Graph Builder and Graph Grammar package are currently implemented in Klone, a Common Lisp dialect. Advantages of Klone were useful to quickly develop these experimental packages on graph grammar. Graph Builder and Graph Grammar are about 7000 lines of Klone. All run under UNIX system.

6 Conclusion

We have presented a method and a system to recognize scanned mathematical formulas. The system is composed of three clearly separated modules (OCR, graph builder, graph grammars and parsing). On a theoretical level, we use a graph grammar and we have define a method to remove ambiguities of grammars. On a practical level, we have a first implementation of the method, which works on various complex formulas, 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, vertical operators, 2D tree-operators, 2D cyclic-operators, implicit operators. For most grammars dealing with these constructions, we are able to remove correctly ambiguities, with the presented criterions. Just some cases need heuristics to solve.

In future, we will focus on the two first parts of *Ofr*: the OCR component, and the graph builder. The main problem in graph building is to find a good tradeoff between two extreme cases: a graph with 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.

References

[1] R. H. Anderson, "Syntax Directed Recognition of Hand-Printed Two-Dimensional Mathematics," *Interactive Systems for Experimental Applied Mathematics*, pp. 436-459, 1968.

[2] H. Bunke, "Graph Grammars as a Generative Tool in Image Understanding," *Graph Grammars and their Application to Computer Science*, Vol. 153, pp. 8-19, 1982.

[3] P. Chou, "Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar," *Proc. SPIE Conf. on Visual Communications and Image Processing IV*, pp. 852-863, 1989.

[4] H. Fahmy and D. Blostein, "A Graph Grammar for High-Level Recognition of Music Notation," *Proc. of 1st ICDAR (International Conference on Document Analysis and Recognition)*, Vol.1, pp. 70-78, 1991.

[5] H. Fahmy and D. Blostein, "A Survey of Graph Grammars: Theory and Applications," *Proc. of the 11th ICPR (International Conference on Pattern Recognition)*, 1992.

[6] R. J. Fateman, T. Tokuyasu, B. P. Berman, and N. Mitchell, "Optical Character Recognition and Parsing of Typeset Mathematics," *Journal of Visual Communication and Image Representation*, Vol. 7, 1995.

[7] S. Lavirotte and L. Pottier "Graph Grammar for Mathematical Formula Recognition" *Technical Report* INRIA, 1997.

[8] H.-J. Lee and M.-C. Lee, "Understanding Mathematical Expressions in a Printed Document," *Proc. of the 2nd ICDAR*, pp. 502-505, 1993.

[9] M. Nagl, "A Tutorial and Bibliographical Survey on Graph Grammars," *Workshop on Graph-Grammars and their Application to Computer Science and Biology*, pp. 70-126, 1978.

[10] M. Okamoto and A. Miyazawa, "An Experimental Implementation of a Document Recognition System for Papers Containing Mathematical Expressions," *Structured Document Image Analysis*, pp. 36-53, 1992.

[11] J. Pfaltz and A. Rosenfeld, "Web Grammars," *Proc. 1st International Joint Conference on Artificial Intelligence*, pp. 609-619, 1969.

[12] J.-C. Raoult and F. Voisin. "Set-Theoretic Graph Rewriting," *Technical report* IRISA, 1992.

[13] H. M. Twaakyondo and M. Okamoto, "Structure Analysis and Recognition of Mathematical Expressions," *Proc. of 3rd ICDAR*, pp. 430-437, 1995.

[14] Z.-X. Wang and C. Faure, "Structural Analysis of Handwritten Mathematical Expressions," *Proc. of the 9th ICPR*, pp 32-34, 1988.