Off-line Recognition of Printed Mathematical Expressions Using Stochastic Context-Free Grammars

IARFID Master Thesis
Universidad Politécnica de Valencia
September 10, 2010

Author: *Francisco Álvaro Muñoz*
Advisor: *Joan Andreu Sánchez Peiró*

Contents

Chapter 1

Preliminaries

1.1 Introduction

The purpose of this work is study the off-line recognition of printed mathematical expressions. Every year a lot of technical papers that contain many information are published, and the mathematical expressions are an important part of them. Furthermore, the tactile interfaces are being incorporated more and more to the new devices, which allow us to use a pen as a computer input. For that reason, it is very interesting to be able to process all this information automatically.

Automatic recognition of mathematical expressions requires to solve many issues, and it is a hard task because it is necessary to jointly perform segmentation, symbol recognition and structural analysis. In this work, we study several methods to deal with the recognition of mathematical symbols and the use of formal grammars to model the structure of mathematical expressions.

Document Image Analysis [25] is an area of great interest in the pattern analysis research community because it comprises several pattern recognition problems. The digitization process of scientific documents requires to perform specific layout analysis [26, 17] and recognition processes for each region of interest. These regions can contain text, images or mathematical expressions among others. In this case, mathematical expression recognition is necessary to properly handle this type of information.

Tactile or pen-based interfaces are becoming more common lately, and as a result, development of applications that can work with this kind of input is increasing. This way of obtaining the data produces more different representations of mathematical expressions than when they are obtained from scanned documents. Depending on the source and how the data is obtained, a mathematical expression can be on-line/off-line processed and/or printed/handwritten processed. Each mode has its own properties and issues that are described in sections 1.2.1 and 1.2.2.

Mathematical expression recognition has been studied since many years ago [7, 5], but it is only until recently that this problem has attracted more attention from the research community [24, 4]. Lately, new devices provide several ways to introduce information to computer systems, as tactile or pen-based interfaces. Hence, interest in developing applications that work with this type of input has been growing significantly during last years. Consequently, on-line handwritten mathematical expression recognition is the most researched mode [9, 35, 11, 34]. However, the off-line case has been less explored [23, 28], but it is a very interesting issue for many applications, as document recognition or information retrieval, among others.

Mathematical expression recognition is a difficult task, which in turn involves three main problems to solve [11]: segmentation, symbol recognition and structural analysis. There are some works that deal with the segmentation problem directly [19, 32], whereas other works handle this problem integrated with other stages of the whole problem [23]. The mathematical symbol recognition is tackled as a classification task, and several methods have been described in the literature, like SVM [18] or distance-based classifiers [30]. Finally, the structural analysis step is very a challenging issue in the mathematical expression recognition problem. Several approaches have been considered to solve this problem, such as using formal grammars [7], tree transformation [36], Hidden Markov Models [13] or computing the minimum spanning tree [29].

Generally, the mathematical expression recognition problem is tackled by systems that deal with all the steps integrated. In this way, the structural analysis stage makes the decisions to process the input expression successfully, using the structural information to correctly solve the segmentation and symbol recognition tasks.

1.2 Handling Mathematical Expressions

Recognition of mathematical expression is an easy task for human beings, but to recognize mathematical expressions by a computer is not a trivial task. Currently, when the user wants to enter some mathematical information to the computer, he usually needs to know a *professional editor* that can understand a specific syntax.

Thus, LaTeX is a document preparation system that provides a powerful way for writing mathematical expressions by using a special syntax. It is the *de facto* standard for the communication and publication of scientific documents. LaTeX is a complete high-quality typesetting system which includes features for the production of technical documents. The LaTeX syntax is simple and powerful. For example, the following code:

```
x^2 + \sum_{i=0}^n \beta_i = \sqrt{ \pi + \frac{x}{n} }
```

produces the expression

$$\boxed{x^2 + \sum_{i=0}^{n} \beta_i = \sqrt{\pi + \frac{x}{n}}} \, .$$

But the natural way for human beings of producing or reading scientific documents that include mathematical expressions is writing or reading directly the *image* of the expression. In this way recognition of mathematical expressions becomes a very interesting problem that requires an adequate solution. Given the different ways in which humans produce reading documents, mathematical expression recognition can be considered from different point of views. In the following sections we summarize these point of views.

1.2.1 Printed versus Handwritten

As in text recognition, the samples could be printed or handwritten. Obviously, printed expressions are more regular and constrained than the handwritten approach. For that reason, the printed recognition problem is considered to be easier than the handwritten case (see Figure 1.1). This is because different people have different handwriting styles, in addition to the variability characteristic of the human skills.

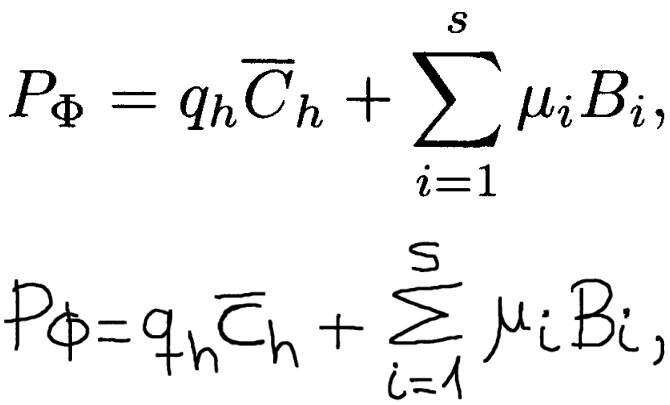


Figure 1.1: Example of expression in printed and handwritten mode.

1.2.2 On-line versus Off-line

A mathematical expression could be represented in several ways. As in many other fields like text recognition, mathematical expression recognition can be divided into two different problems [22]:

- *Off-line*: it involves the automatic recognition of a mathematical expression given as an image. The samples does not contain any time information (Figure 1.2 up).

- *On-line*: it involves the automatic recognition of a mathematical expression given as a sequence of points that describes a path in the space (Figure 1.2 down).

The on-line approach is usually related with the handwriting case, because this type of information is easily obtained using tactile interfaces. In addition, it is possible to obtain an off-line sample from an on-line sample, but there is not satisfactory solutions to perform the opposite transformation. This difference in the data representations produces that the methods used to solve the mathematical recognition problem are different. This is most noticeable in the segmentation and symbol recognition process, because the nature of the input is significantly different, while structural analysis is conceptually more similar.

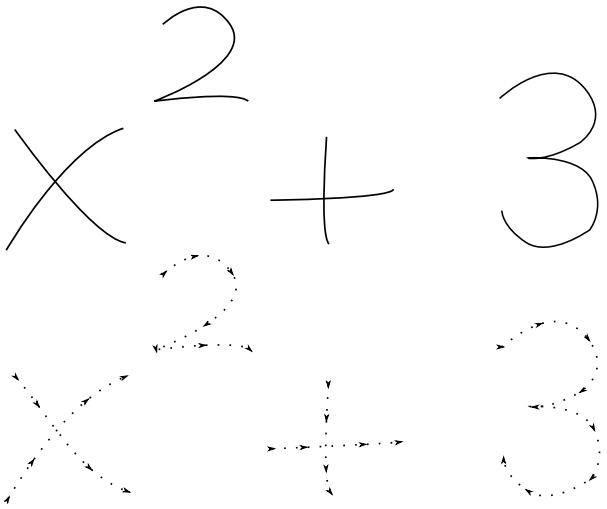


Figure 1.2: Example of expression in off-line and on-line mode.

1.3 Problem Statement for this Master Thesis

In this work we will focus on off-line recognition of printed mathematical expression. This problem has been studied in few works [36, 28, 23], and the current results, specially structural results, suggest that there is room for improvements.

In addition to the recognition problem itself, an adequate solution of this problem will be useful for innovative applications like:

- To search documents that have a given expression
- To help to the speech transcriptions/recognition of recorded lectures that include slide images by providing a vocabulary

In order to tackle the problem of recognition of printed mathematical expression, in this work we will use formal grammars to represent the structural relations among the mathematical symbols. As in most pattern recognition scenarios, we have to solve two main problems: the automatic learning of the models and the recognition problem through an interpretation process. In this work we will focus specifically in the interpretation process, and we propose the learning process for future work.

Context free models will be used for the interpretation process. These models are considered appropriate to represent the two-dimensional structural constraints that are present in mathematical expressions. In the following sections we will review some notation and some classical algorithms related to those that we will introduce through this work.

1.4 Grammars

Formal grammars are used in this work to model mathematical expressions. A formal grammar can be defined as a tuple

$$G = (N, T, P, S)$$

where

- **N:** Finite set of nonterminal symbols.
- **T:** Finite set of terminal symbols ($N \cap T = \emptyset$).
- **P:** Set of derivation rules $\alpha \to \beta$ with $\alpha \in V^*NV^*$ and $\beta \in V^*$, such that $V = N \cup T$.
- **S:** Starting symbol ($S \in N$).

Given a sequence of symbols and a grammar G, we define the derivation process as

$$\mu\alpha\delta \overset{*}{\Rightarrow} \mu\beta\delta \quad \text{iff} \quad \exists(\alpha \to \beta) \in P; \quad \mu, \delta \in V^*.$$

A formal grammar is a precise definition of a formal language, which in turn is a set of strings over some alphabet T. The formal language generated by a grammar G is the set

$$L(G) = \{x \in T^* \mid S \overset{*}{\Rightarrow} x\}$$

Formal grammars can be classified in four groups of progressively restricted grammars, which is known as *Chomsky Hierarchy*. The grammars less restricted are more powerful and expressive. However, the computational complexity of the algorithms that handle each type of grammar is lower when working with more constrained grammars. In this work we are interested in *context-free grammars*, because they are expressive enough to model mathematical expressions and they are also computational tractable. The production rules of this type of grammars are restricted to take the form $\alpha \to \beta$ with $\alpha \in N$ and $\beta \in (N \cup T)^*$.

Context-Free Grammars (CFG) are often defined in *Chomsky Normal Form* (CNF), which means that the production rules are of the form $A \to BC$ or $A \to t$, where $A, B \in N$ and $t \in T$. Every grammar in CNF is context-free, and conversely, every CFG can be transformed into an equivalent one which is in CNF.

A *Stochastic Context-Free Grammar* (SCFG) is a CFG in which each production rule is augmented with a probability. So, every rule r_i has associated a probability $Pr(r_i) = Pr(A \to \alpha) \in]0, 1]$, and the following constraint must be accomplished:

$$\sum_{\forall \alpha_j} Pr(A \to \alpha_j) = 1.$$

In other words, the probability of all the rules having the same left-hand nonterminal must sum one.

The probability of a derivation d_x for a given string x produced by a grammar G is the product of the probabilities of the applied derivation rules

$$Pr_G(x, d_x) = \prod_{r_i \in d_x} Pr(r_i).$$

The Cocke-Younger-Kasami (CYK) algorithm [1] determines whether a string can be generated by a given CFG and, if so, how it can be generated. The CYK algorithm is a dynamic programming algorithm that builds increasing size problems from combinations of lower size subproblems (bottom-up parsing). The classical version of the CYK algorithm just checks if a given string is generated by a grammar. A stochastic version of the CYK algorithm (see Figure 1.3) computes the probability of most probable derivation by storing the probabilities of the partial parsed subproblems, and it allows us to compute:

$$\widehat{Pr}_G(x) = \max_{\forall d_x} Pr_G(x, d_x).$$

It is important to note that the time complexity of CYK algorithm is $O(n^3)$ where n is the size of the string to be parsed.

This stochastic version of the CYK algorithm is presented because in order to deal with the mathematical expression recognition problem, a two-dimensional (2D) extension of this algorithm is needed. This 2D extension of the CYK algorithm is explained in section 1.5.

Input: $G = (N, T, P, S)$, $G_s = (G, Pr)$ in CNF and $x = x_1 x_2 \dots x_n \in T^*$
Output: $\widehat{Pr}_G(x)$ (if $x \notin L(G)$ then $\widehat{Pr}_G(x) = 0.0$)

Method
for all i, j, A **do**
$t[i, j, A] = 0.0$

for all $i = 0 \dots n-1$ **do**
for all $(A \to x_i) \in P$ **do**
$t[i, i+1, A] := Pr(A \to x_i)$

for all $j = 2 \dots n$ **do**
for all $i = 0 \dots n-j$ **do**
for all $i = 0 \dots n-j$ **do**
for all $(A \to BC) \in P$ **do**
$prob := t[i, i+k, B] \cdot t[i+k, k+j, C] \cdot Pr(A \to BC)$
if $prob > t[i, i+j, A]$ **then** $t[i, i+j, A] := prob$

return $t[0, n, S]$
End method

Figure 1.3: CYK algorithm for Stochastic CFG

1.5 Two-dimensional Extension of CF Parsing

SCFG are a powerful formalism of syntactic pattern recognition that has been extensively used for string patterns. However, it is possible to lightly modify this formalism so that grammars can model 2D problems. In this work, we are interested in modeling mathematical expressions using SCFG. For this reason, a 2D extension is introduced in a similar way as in [35].

There are mainly two differences from a SCFG. First, in the 2D case, terminal and nonterminal symbols describe *regions* instead of symbols. This means that terminal and nonterminal symbols of the grammar contain some features like 2D coordinates, and others. Second, the production rules have an additional parameter that describes the spatial relation among the symbols. Formally, a 2D SCFG is a standard SCFG in which the production rules are as follows:

$$A \xrightarrow{spr} \alpha$$

where $A \in N$; $\alpha \in (N \cap T)^*$ and *spr* denotes the spatial relation that models the rule. Common spatial relations for mathematical expression recognition

are: *horizontal*, *vertical*, *inside*, *subscript* and *superscript*. The probability of a rule is:

$$Pr(A \xrightarrow{spr} \alpha) = Pr(\alpha \mid A, spr).$$

Finally, this models can be represented in CNF (see 1.4) and the rules are as follows:

$$A \to t$$

$$A \xrightarrow{spr} BC$$

$$A, B, C \in N, \quad t \in T.$$

The terminal productions do not contain the spatial relation because there is no spatial relation with only one symbol. Their probabilities are

$$Pr(A \to t) = Pr(t \mid A)$$
$$Pr(A \xrightarrow{spr} BC) = Pr(B, C \mid A, spr).$$

In conclusion, the SCFG 2D extension is achieved by adding a spatial relation model in the binary production rules. Moreover, the symbols of the grammar contains attributes to provide 2D information needed at the process of providing a parse tree.

In order to illustrate the SCFG 2D extension we present a simple grammar G_{exp} that models addition and subtraction of integer fractions:

$$G = (N, T, P, S); \quad G_{exp} = (G, Pr, spr)$$

$$\begin{aligned} N &= \{Exp, OpExp, OvNum, Num, Over, Op\} \\ T &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, hline, +, -\} \\ S &= Exp \\ spr &= \{horizontal, vertical\} \end{aligned}$$

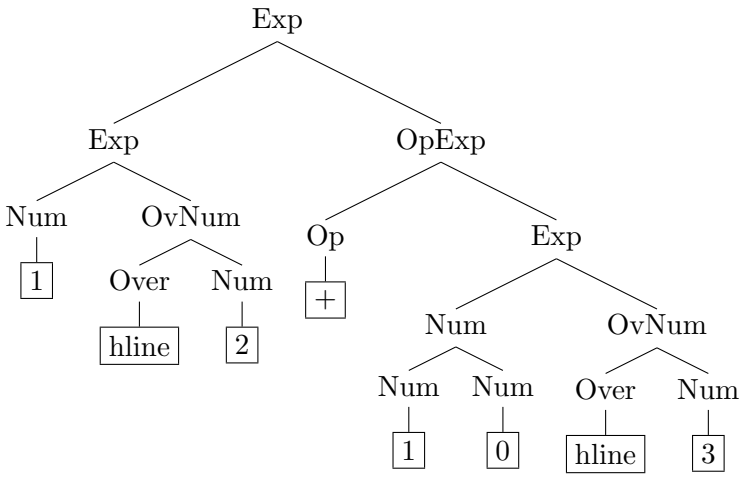
where the production rules P are

Exp	$\xrightarrow{horizontal}$	Exp	OpExp
Exp	$\xrightarrow{vertical}$	Num	OvNum
OvNum	$\xrightarrow{vertical}$	Over	Num
OpExp	$\xrightarrow{horizontal}$	Op	Exp
Num	$\xrightarrow{horizontal}$	Num	Num
Num	$\longrightarrow$	[0,1,2,3,4,5,6,7,8,9]	
Over	$\longrightarrow$	[hline]	
Op	$\longrightarrow$	[+,-]	

and Pr is a probability distribution that properly models the terminals recognition and spatial relations among regions (Sections 2.3 and 3.3 explain how probabilities are computed, and a complete example is presented in Section 3.4). For example, the expression

$$\frac{1}{2} + \frac{10}{3}$$

is generated by the previous grammar as is represented in the following derivation tree



Chapter 2

Recognition Steps

Usually the problem of mathematical expression recognition is divided into three steps: *segmentation*, *symbol recognition* and *structural analysis*. In this chapter we present which methods are studied in this work to solve these major subproblems independently.

2.1 Segmentation

Given a binary image which contains the representation of a mathematical expression, the first step is to segment this image into groups. The goal is that each of these groups form exactly one symbol.

In this work, the method chosen to solve the segmentation problem is to compute the connected components of the input image. It can be easily carried out by considering the black pixels to be 8-connected. In other words, pixels are neighbors to every pixel that touches one of their edges or corners. Figure 2.1 shows an example of the 8-connected components computed for a mathematical expression image.

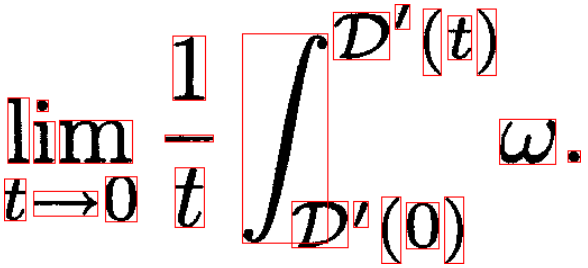


Figure 2.1: 8-connected components of an binary image.

There are many mathematical symbols which are composed by more than one connected component, but this problem will be tackled in other stages of the expression recognition process and it will be explained later.

2.2 Symbol Recognition

After the segmentation step, we have a list of regions that can contain a mathematical symbol. In order to make a decision, a mathematical symbol classifier is needed. In this work we tried several techniques to solve this problem.

Recognition of typeset mathematical symbols is a difficult problem due to several reasons: first, there is a large number of different symbols; second, the number of font-types can be different in the same mathematical expression (e.g., roman, italic, and calligraphic); and third, symbols can be of different size in the same expression. Several techniques have been proposed for the off-line recognition of printed mathematical symbols and a good review can be seen in [10]. A combination of classifiers was proposed in [10] that achieved very good results for a large database. However, a comparison of several techniques on the same database would be interesting. In this work we compared several classification techniques for recognition of printed mathematical symbols that proved to be very efficient for other classification tasks.

In pattern recognition, the representation of the problem is important to the classifier performance. In this case, given a region of an image that could contain a mathematical symbol, it was normalized to a fixed size of $n \times m$. After that, a vector of $n \cdot m$ components was formed by concatenating the rows of the normalized region, and this vector was the representation used for each symbol in the classification task.

The HMM models has a difference in the representation of the symbols. The HMM expects a sequence of vectors as input, for that reason the image was transformed into a sequence of fixed-dimension feature vectors. In other words, the image was adequately normalized to a fixed size rows, keeping the aspect ratio, and then a sequence of vectors was obtained.

2.2.1 Nearest Neighbor

The k-Nearest-Neighbor (k-NN) rule is a very popular pattern classification rule that provides good results when the number of prototypes is large. This is a usual classification technique that has also been tested for mathematical symbols classification [9]. This classifier doesn't need to be trained, because only it is necessary to have available a set of labeled samples. Given the vector representation of a mathematical symbol, each sample is interpreted as a point in a high-dimensional space. Therefore, given a test sample, the distance is computed to all the prototypes of the labeled set. Then, the k-NN classifier uses the k nearest prototypes to the test sample to determine its class, which is the most voted.

2.2.2 Support Vector Machines

Support vector machine (SVM) is a maximum margin classifier that has demonstrated to be a powerful formalism for recognition tasks. In this work we used the multi-class SVM described in [8]. This technique has been previously used for mathematical symbol recognition with successful results.

For the SVM classification, we used the $\text{SVM}^{multiclass}$ software with default options[1]. In experiments reported in [18] on mathematical symbol classification with the INFTY corpus, a linear kernel obtained better classification results than a Gaussian kernel and a cubic polynomial kernel. Therefore, a linear kernel function was just used.

2.2.3 Weighted Nearest Neighbor

The Weighted Nearest Neighbor (WNN) technique is an improvement of the classical 1-NN [20]. A discriminative technique is used to learn a weighted distance by using the 1-NN rule with a training set. A distance weighting scheme is proposed which can independently emphasize prototypes and/or features. Several alternatives are considered in [20]: using a different weight for each prototype, using a different weight for each class and characteristic, or using a combination of the previous alternatives. In this work, we used the last alternative previously mentioned, that is, a different weight for each prototype combined with a different weight for each class and feature. The reason for this was that in a training set there are samples more representative than others, and also in symbol representation the importance of each pixel is different. Consequently, it is reasonable to weight both the prototypes and the features for each class.

2.2.4 Hidden Markov Models

Hidden Markov Models (HMM) have been widely used for mathematical symbol classification in on-line mathematical expression recognition [9]. However their use in off-line recognition remains unexplored. In recent years, HMM has been successfully used for off-line handwritten text recognition [31]. In this work, we explore the technique described in [31] applied to printed mathematical symbol recognition.

We have slightly adapted those techniques for printed mathematical symbol recognition. Given the novelty of this approach for this recognition task, we explain it in more detail (see [31] for additional details). In the preprocessing stage, noise reduction is carried out in the symbol image and then, it is adequately normalized to a fixed size row, keeping the aspect ratio (see Figure 2.2.a). The image is transformed into a sequence of fixed-dimension feature vectors as follows: the image is divided into a grid of small square

[1] `http://svmlight.joachims.org/svm_multiclass.html`

cells, sized a small fraction of the image height. Each cell is characterized by the following features: normalized grey level (see Figure 2.2.b), horizon-

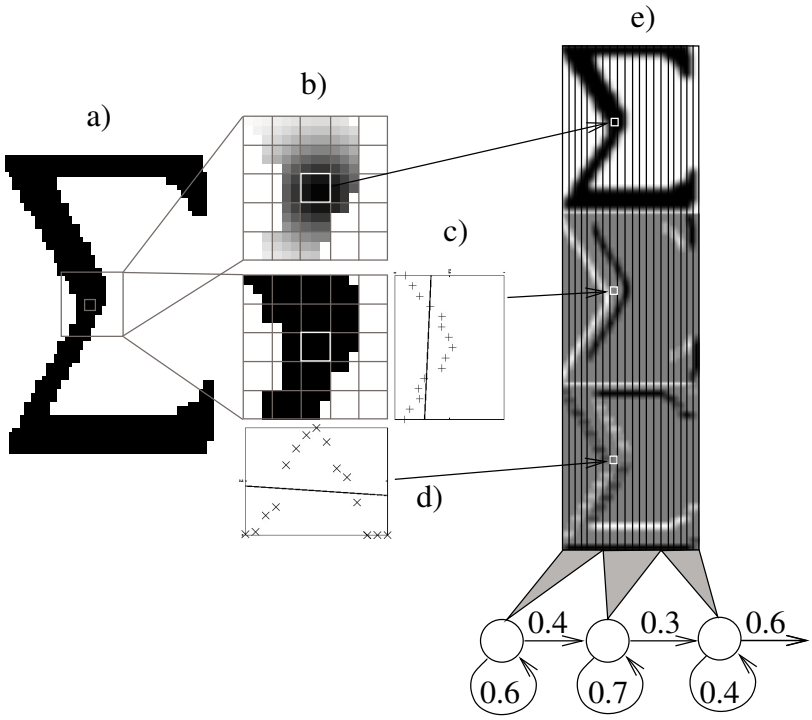


Figure 2.2: Feature extraction for HMM recognition.

tal grey-level derivative (c) and vertical grey-level derivative (d). To obtain smoothed values of these features, feature extraction is extended to a 5×5 cell window centered at the current cell and weighted by a two-dimensional Gaussian function in b) and a unidimensional Gaussian function in c) and d). The derivatives are computed by least squares fitting a linear function. Columns of cells are processed from left to right and a feature vector is built for each column by stacking the features computed in its constituent cells. Figure 2.2.e shows a graphical representation of the obtained values.

Finally, each symbol is represented by a sequence of feature vectors, which are used to train a HMM for each class, and at the classification stage one sample is labeled to the class of the HMM with higher probability.

2.3 Structural Analysis

Structural analysis is a challenging task in the recognition of mathematical expressions. From the segmentation and symbol recognition steps, the problem is to determine the relations among these symbols in order to build a

complete structure. This structure represents the mathematical expression recognized.

Such as we commented in Section 1.1 several approaches have been tried to solve the structural analysis problem [7, 36, 13, 29]. In this work we used SCFG to model the mathematical expressions structure and the CYK algorithm to parse the input sample to obtain the most probable derivation. The 2D extension of SCFG was explained in Section 1.5, but algorithm must be modified to work with this type of formal grammars. Given the SCFG in CNF, the probabilities are formally defined as

$$Pr(A \to t) = Pr(t \mid A)$$
$$Pr(A \xrightarrow{spr} BC) = Pr(B, C \mid A, spr).$$

In a similar way as in [35], we determine these probabilities using probability functions. On the one hand, the $Pr(t \mid A)$ probability is obtained from the mathematical symbol classifier as the probability that region t belongs to class c such that $c \in (A \to c)$. On the other hand, the $Pr(B, C \mid A, spr)$ probability models the spatial relation spr between B and C regions, so the probability is computed as

$$Pr(B, C \mid A, spr) = Pr(B)Pr(C)Pr(B, C \mid spr)$$

where $Pr(B)$ and $Pr(C)$ are obtained from the CYK parsing table and $Pr(B, C \mid spr)$ represents the probability that regions B and C are spatially arranged according to spr (more details in Section 3.3.2). The modified algorithm is shown in Figure 2.3. This algorithm has two special operations that are defined below. Given two regions c_1 and c_2, the $\oplus$ operator computes the smallest rectangle containing both regions, and the $\uplus$ operator is defined as follows.

Let S be a set of elements $(A_i, reg_i, prob_i)$, such that A_i is a nonterminal symbol, reg_i is a region in the image, and $prob_i$ is its probability, and let x be a new element $(B, reg', prob')$. The operation $S \uplus x$ is defined as:

if $\exists y = (A, reg, prob) \in S$ **such that** $A = B$ **and** $reg = reg'$ {
  if $prob > prob'$ **then** $discard(x)$
  else replace y with x
}
else $S = S \cup x$

The idea is that each region of the mathematical expression can be parsed in several ways, but the probability is maximized.

Looking at the 2D CYK algorithm, the first remarkable difference is that the parsing table is indexed by only one value. On the standard CYK parsing the two indexes explain the positions that define some substring. In the 2D case, there is a table level for each subproblem size, and these levels contains

Input: $Gs = (N, T, P, S, Pr)$ in CNF and $x = \{x_1, x_2, \ldots, x_n\} \in T^*$
Output: $\widehat{Pr}_G(x)$ (if $x \notin L(G)$ then $\widehat{Pr}_G(x) = 0.0$)

Method
for all $i = 0 \ldots n-1$ **do**
for all $(A \to x_i) \in P$ **do**
if $Pr(A \to x_i) > 0.0$ **then** $t[1] := t[1] \cup (A, x_i, Pr(A \to x_i))$

for all $j = 2 \ldots n$ **do**
for all $a = 1 \ldots n-1$ **do**
for all $c_1 \in t[a]$ **do**
for all $c_2 \in t[n-a]$ **do**
for all $(A \xrightarrow{spr} BC) \in P$ **do**
$prob := Pr(c_1 \,|\, B) \cdot Pr(c_2 \,|\, C) \cdot Pr(c_1, c_2 \,|\, spr)$
if $prob > 0.0$ **then** $t[j] := t[j] \uplus (A, c_1 \oplus c_2, prob)$

return $t[0, n, S]$
End method

Figure 2.3: CYK Algorithm for 2D SCFG.

a set of elements which contains their two-dimensional space information. For that reason, at the initialization loop the built subproblems are added at $t[1]$ level, that is to say that they cover one input symbol. After that, the parsing process continues by building new subproblems of increasing size, where the spatial relation model contributes to the probability of each possibility.

Finally, the time complexity of the algorithm is $O(n^4|P|)$ whereas the time complexity of the classical CYK is $O(n^3|P|)$. This is because there are four loops over the n regions (j, a, c_1 and c_2 variables), and an additional loop for the production rules P. However, in section 3.3.1 this complexity is discussed and reduced.

Chapter 3

Developed System

This chapter describes the system developed for parsing mathematical expressions. This system uses a two-dimensional *Stochastic Context-Free Grammar* (SCFG) to model spatial relations between symbols. The parsing is carried out with the CYK algorithm that was described in previous sections.

3.1 Mathematical Expressions Grammar

In order to carry out the parsing of the mathematical expressions, a 2D SCFG grammar is needed. In this work we did not study the automatic learning of a model from a data set. Thus, this grammar was defined manually trying to cover a wide range of expressions. The LaTeX syntax is very permissive, so a slightly constrained grammar was used. We tried to model all the mathematical expressions that appeared in the data sets that we used in the experiments. The grammar parsed most of mathematical expressions, but there were cases that were not modeled, for example left subscripts or superscripts (${}_1^2a$) or matrices. Figure 3.1 shows the binary rules of the 2D grammar that we defined, where production rules are equiprobable and an extra field was added to each rule to obtain a formatted output. This information is represented at the last column of each production rule and it is explained below. For example, the binary rule

$$\mathsf{OverExp} \xrightarrow{\textit{Vertical}} \mathsf{Over}\quad \mathsf{Exp}\quad \texttt{"\$2"}$$

represents that a *OverExp* region can be obtained by the combination of two regions *Over* and *Exp* where their spatial relation implies that the former is above the second.

Terminal production rules are not shown in Figure 3.1, because there are a large number of terminals. The nonterminals *Exp, Auxh, Auxs, Let, Over, BigOp, OpUn, OpBin, OverSym, UnderSym, LeftPar, RightPar* and *Sqrt*, have terminal production rules. For example, the *OpUn* nonterminal has associated the productions of the symbols $\exists$, $\forall$, $-$, $\neg$, $+$ and $\pm$, which represents unary operators.

Exp	$\xrightarrow{Horizontal}$	Exp	Auxh	`"$1 $2"`
Exp	$\xrightarrow{Horizontal}$	Auxh	Exp	`"$1 $2"`
Exp	$\xrightarrow{Horizontal}$	Auxh	Auxh	`"$1 $2"`
Exp	$\xrightarrow{SuperScript}$	Exp	Auxs	`"{$1}^{$2}"`
Exp	$\xrightarrow{SubScript}$	Exp	Auxs	`"{$1}_{$2}"`
Exp	$\xrightarrow{SubScript}$	BigOp	Exp	`"$1_{$2}"`
Exp	$\xrightarrow{Horizontal}$	Exp	Exp	`"$1 $2"`
Exp	$\xrightarrow{Horizontal}$	OpUn	Exp	`"$1$2"`
Exp	$\xrightarrow{Horizontal}$	Exp	OBExp	`"$1 $2"`
OBExp	$\xrightarrow{Horizontal}$	OpBin	Exp	`"$1 $2"`
Exp	$\xrightarrow{SuperScript}$	Exp	Exp	`"{$1}^{$2}"`
Exp	$\xrightarrow{SubScript}$	Exp	Exp	`"{$1}_{$2}"`
Exp	$\xrightarrow{Vertical}$	Exp	OverExp	`"\frac{$1}{$2}"`
OverExp	$\xrightarrow{Vertical}$	Over	Exp	`"$2"`
Exp	$\xrightarrow{VerticalStrict}$	OverSym	Exp	`"$1{$2}"`
Exp	$\xrightarrow{VerticalStrict}$	Exp	UnderSym	`"$2{$1}"`
Exp	$\xrightarrow{Horizontal}$	LeftPar	RPExp	`"$1 $2"`
RPExp	$\xrightarrow{Horizontal}$	Exp	RightPar	`"$1 $2"`
Exp	$\xrightarrow{Horizontal}$	Exp	SSExp	`"{$1}$2"`
Exp	$\xrightarrow{Horizontal}$	BigOp	SSExp	`"$1$2"`
SSExp	$\xrightarrow{SupSubScript}$	Exp	Exp	`"_{$2}^{$1}"`
SSExp	$\xrightarrow{SupSubScript}$	Auxs	Exp	`"_{$2}^{$1}"`
SSExp	$\xrightarrow{SupSubScript}$	Exp	Auxs	`"_{$2}^{$1}"`
SSExp	$\xrightarrow{SupSubScript}$	Auxs	Auxs	`"_{$2}^{$1}"`
Exp	$\xrightarrow{Vertical}$	Exp	BigOpExp	`"$2^{$1}"`
BigOpExp	$\xrightarrow{Vertical}$	BigOp	Exp	`"$1_{$2}"`
Exp	$\xrightarrow{Horizontal}$	BigOpExp	Exp	`"$1 $2"`
Exp	$\xrightarrow{Inside}$	Sqrt	Exp	`"\sqrt{$2}"`
Exp	$\xrightarrow{Horizontal}$	Exp	Func	`"$1 $2"`
Exp	$\xrightarrow{Horizontal}$	Func	Exp	`"$1 $2"`
Func	$\xrightarrow{Horizontal}$	Let	2Let	`"{$1$2}"`
Func	$\xrightarrow{Horizontal}$	2Let	2Let	`"{$1$2}"`
2Let	$\xrightarrow{Horizontal}$	Let	Let	`"$1$2"`
Exp	$\xrightarrow{Vertical}$	Func	Exp	`"$1_{$2}"`

Figure 3.1: Mathematical expressions 2D grammar.

3.2 CYK Table Initialization

First, we explain the initialization step in the CYK algorithm as it was introduced in Section 1.4 and Section 2.3.

3.2.1 Segmentation and Symbol Recognition

Given an image of a mathematical expression, first, the connected components are calculated as described in Section 2.1. As a result, a set of image regions is obtained. For all of these components, a mathematical symbol classifier is used to determine the class of each one. This task could be carried out by any of the classifiers explained in Section 2.2. In our case the *Nearest Neighbor* (NN) was chosen, because it is simple an it achieved good results as we explain in the experiments.

One mathematical symbol can belong to multiple classes due to several kind of misclassification. For example, the + symbol can be interpreted as a binary operator (1+2), unary operator ($+b$) or a standard symbol (Σ^+). The − symbol can be interpreted as a minus symbol, as an overline, or as the bar of a fraction, among other. For that reason, the symbol recognition process classifies each terminal in several nonterminals that represent its possible interpretations. Thus, the symbol + is stored in the CYK table with its probability for each nonterminal *OpUn, OpBin* and *Auxs*. Generally, each region is classified in all terminal symbols and each nonterminal symbol associated to the corresponding rule is added to the parsing table. Finally, the parsing process will decide the most probable interpretation taking into account the expression structure.

The NN classifier computes the Euclidean distance between vectors, but in the CYK algorithm probabilities are needed. Formally, given an image x, let $\hat{p}_c$ be the nearest prototype of class c from a labeled set, and let $d(x, \hat{p}_c)$ be the distance between them. The probability of x to belong to the class c can be obtained as

$$p(x \mid c) \propto e^{-d^2(x,\hat{p}_c)}$$

So, the probability was proportional to this expression. In the implementation the final expression used as

$$p(x \mid c) = e^{-d^2(x,\hat{p}_c)/F} = \left(e^{-d^2(x,\hat{p}_c)}\right)^{\frac{1}{F}}$$

due to scaling problems, where F is a positive integer number.

Finally, the pseudocode for the segmentation and symbol recognition that initializes the CYK table is shown in Figure 3.2. This code shows how each region is labeled in several classes. We used a threshold to avoid exploring improbable hypothesis. This was done in order to limit the search space, but in this way we could not guarantee that the optimal solution was achieved.

```
for all connected components C of image X do {
    for all nonterminals A of terminal productions (A → t) do {
        (class, prob) =NN(C, A)    //Symbol Classifier
        if ( prob > threshold ) then
            t[1] := t[1] ∪ (A_class, C, prob)
    }
}
```

Figure 3.2: Segmentation and mathematical symbol classification to initialize the CYK algorithm table.

3.2.2 Multiple Connected Components Detection

It is very common to deal with expressions where some of their symbols are composed of more than one connected component. There are three possible reasons (see examples in Figure 3.3):

- By definition: Some symbols $(=, i, j, ;)$ are composed of more than one connected component, so it is necessary to correctly detect them.
- By noise: Working with real images often means to deal with noise, and these extra pixels are detected as components.
- By degradation: Another problem when working with real images is that they could be degraded, and some symbols could be split on several connected components.

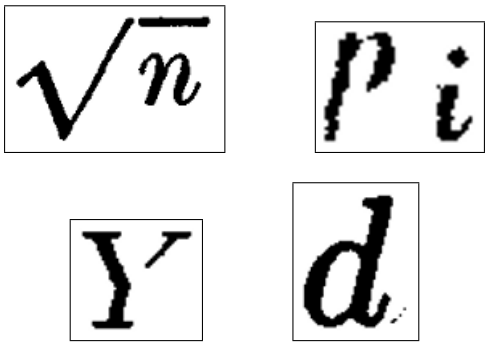


Figure 3.3: Examples of symbols formed by multiple connected components.

A possible way to treat this problem is by merging closer connected components and to get a class and probability from the mathematical symbol classifier. But this introduces a problem, because the probability obtained $Pr(t \mid A)$ is representing a subproblem of size two. However, the probability

of building new subproblems using binary production rules of the grammar is the product of three probabilities

$$Pr(c_1 \,|\, B) \cdot Pr(c_2 \,|\, C) \cdot Pr(c_1, c_2 \,|\, spr)$$

as shown in Section 2.3. Consequently, it was necessary to scale the probability in order to not to favor this type of constructions.

Figure 3.4 shows the multiple connected components detection pseudocode. It works in a similar way to the symbol recognition step, with only a few differences. First, several combinations of closer connected components are tried. Second, the probability obtained from the symbol classifier is scaled to solve the unbalanced probability problem. Finally, the probable combinations are added to the CYK algorithm table as subproblems of size two. In this work we did not considered symbols of more than two connected components. This is because it was necessary to integrate them into the model, otherwise the probabilities were of different nature than the obtained by the CYK algorithm and it tended to provide bad results. We will study this problem in future work.

```
for all connected components C of image X do {
    for all connected components D closer to C do {
        region = D ⊕ C
        for all nonterminals A of terminal productions (A → t) do {
            (class, prob) =NN(region, A)    //Symbol Classifier
            if ( prob > threshold ) then
               t[2] := t[2] ∪ (A_class, region, scale(prob) )
        }
    }
}
```

Figure 3.4: Multiple connected components detection in the CYK algorithm.

In conclusion, for symbols that were possible composed by multiple connected components, those caused by noise were tackled with image filters in a previous step. On the other hand, degraded and naturally split symbols were detected as explained above, but the system could not detect properly the symbols divided in more than two connected components.

3.3 CYK Recursive Steps

At this point, the CYK table parsing is initialized as explained in the previous section. Now the CYK algorithm starts building new subproblems using the SCFG described. The 2D CYK was explained in Section 2.3, but there are some practical issues that must be solved.

On the standard CYK algorithm, it is well known which cells (dynamic programming subproblems) are the predecessors of a certain (i, j) cell, because the one-dimensional space is well constrained. Dealing in this way with 2D images could lead us to algorithms with high time complexity, and therefore this relation is relaxed.

Moreover, the probability of a new problem from other two subproblems c_1, c_2 of minor size and a binary rule ($A \xrightarrow{spr} BC$) is computed as

$$Pr(c_1 \,|\, B) \cdot Pr(c_2 \,|\, C) \cdot Pr(c_1, c_2 \,|\, spr).$$

The probabilities $Pr(c \,|\, A)$ are obtained from the CYK parsing table, but the spatial relation probabilities $Pr(c_1, c_2 \,|\, spr)$ must be defined (see Section 3.3.2). The final algorithm implemented is shown in Figure 3.5, where all practical issues are presented. In this algorithm, the *subset* operation is explained in the following and other notation details are explained later.

3.3.1 Building Subproblems

When building a subproblem of size d, the naive solution is to try all the (a, b) size pairs such that $a + b = d$. In other words, for each subproblem of size a combine it with all the subproblems of size b and let the probability spatial distribution and the CYK parsing to get the best parsing. Using this approach, the cost of the algorithm is $O(n^4|P|)$.

But given a subproblem, and a spatial relation, there is a specific area where the related subproblems are distributed, and it is not necessary to check all the combinations (see Figure 3.6). This is very intuitive, but it is necessary to use some data structure to efficiently obtain the subproblems closer to a given area. The developed system sorts the subproblems of the same size according to their starting horizontal coordinate. This is because mathematical expressions grows in this axis, and after the regions are sorted the system is able to find a point in $O(\log n)$ through a binary search. In [23] this task is performed using feature points and *orthogonal range searching*, achieving the same computational complexity.

Given a region and a spatial relation, we do not want to apply the production rules to subproblems that we know they are improbable. Figure 3.6 shows two cases where given a region (*overline* and μ_i), an area of interest is defined (dotted square). For example, given the region that contains the μ_i expression and the spatial relation *horizontal*, the system only would apply the horizontal production rules with the subproblems which overlap the dotted area. The overline symbol over the C is other example of space search for a *vertical* production.

In Section 2.3 the time complexity of the 2D CYK algorithm was discussed, and we saw that the naive solution had a $O(n^4|P|)$ cost. But using the data structure explained and performing the partial sort with a $O(n \log n)$ algorithm, the time complexity is reduced to $O(|P|n^3 \log n)$. This

```
//Initialization
symbol_recognition(x, t[1])
multiple_components(x, t[2])

//Recursive steps
for all j = 2...n do { //n segmented regions
    for all a = 1...n − 1 do {
        for all c1 ∈ t[a] do {
            z_h = subset( t[n − a], c1 , horizontal)
            z_v = subset( t[n − a], c1 , vertical)
            z_i = subset( t[n − a], c1 , inside)
            for all c2 ∈ z_h do {
                for all (A --spr--> BC) ∈ P such that
                spr ∈ {Horizontal,SuperScript,SubScript,SupSubScript} do {
                    prob := Pr(c1 | B) · Pr(c2 | C) · Pr(c1, c2 | spr)
                    if prob > 0.0 then t[j] := t[j] ⊎ (A, c1 ⊕ c2, prob)
                }
            }
            for all c2 ∈ z_v do {
                for all (A --spr--> BC) ∈ P such that
                spr = {Vertical,VerticalStrict} do {
                    prob := Pr(c1 | B) · Pr(c2 | C) · Pr(c1, c2 | spr)
                    if prob > 0.0 then t[j] := t[j] ⊎ (A, c1 ⊕ c2, prob)
                }
            }
            for all c2 ∈ z_i do {
                for all (A --spr--> BC) ∈ P such that spr = {Inside} do {
                    prob := Pr(c1 | B) · Pr(c2 | C) · Pr(c1, c2 | spr)
                    if prob > 0.0 then t[j] := t[j] ⊎ (A, c1 ⊕ c2, prob)
                }
            }
        }
    }
    sort(t[j])
}

//Parsing output
TEXoutput( t )
```

Figure 3.5: Pseudocode of the implemented CYK 2D parsing algorithm.

$$P_\Phi = q_h \overline{C}_h + \sum_{i=1}^{s} \mu_i B_i,$$

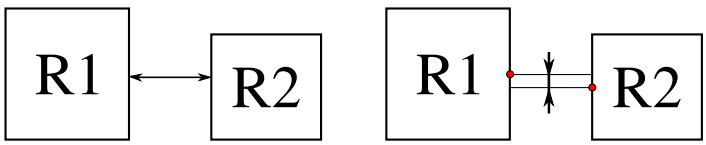
Figure 3.6: Example of searching area for a particular region and spatial relation.

is because the sort is performed only once when a level of a given size is completely built, and the binary search is done for every subproblem of a certain size in $O(\log n)$ (see Figure 3.5). The spatial complexity of the algorithm is $O(n^2|N|)$ because there are n sets with at most n elements, and each one can store several nonterminal symbols $|N|$.

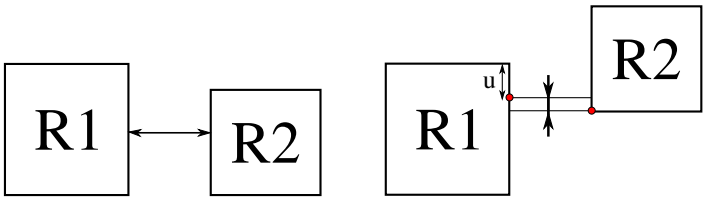
3.3.2 Spatial Relations

There are several spatial relations defined between regions and modeling them properly is a very important point to achieve good results in mathematical expression recognition. Along the parsing process, many regions are combined to form bigger regions, and the features used to describe them were the bounding box coordinates. Using these values, some functions were manually defined to model each spatial relation and to compute the probability $Pr(c_1, c_2 \mid spr)$ used in the parsing algorithm (Figure 3.5). The main issues that these functions took into account are detailed below:

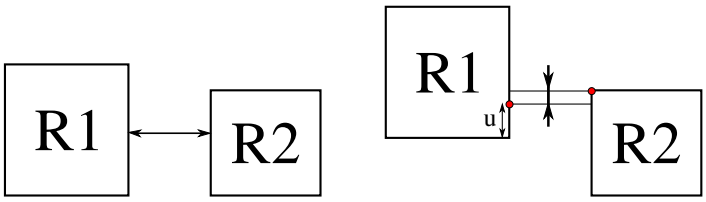
- **Horizontal:** The horizontal distance (left) and the difference between the vertical centers (right).



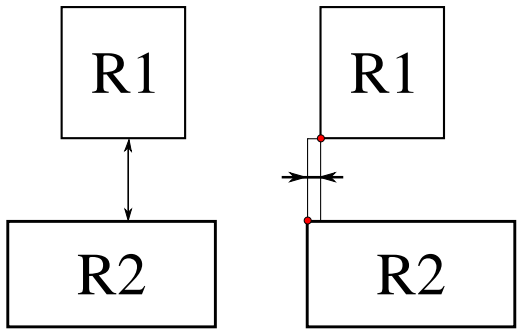
- **Superscript:** The horizontal distance (left) and difference between the bottom left corner of $R2$ and the point at $height(R1) \cdot u$ ($u \in [0, 1]$) of the top right corner of $R1$ (right).



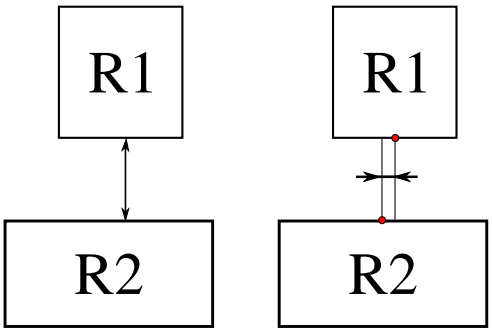
- **Subscript:** The horizontal distance (left) and difference between the top left corner of $R2$ and the point at $height(R1) \cdot u$ ($u \in [0, 1]$) of the bottom right corner of $R1$ (right).



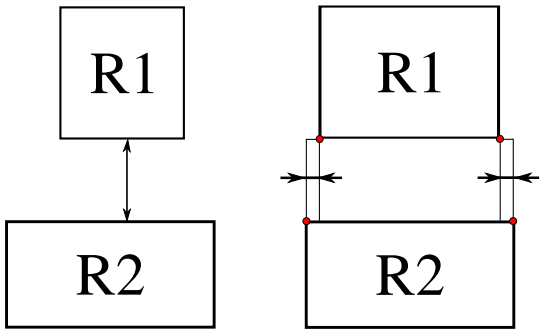
- **SupSubScript:** The vertical distance (left) and the difference between the left horizontal coordinates of both regions (right).



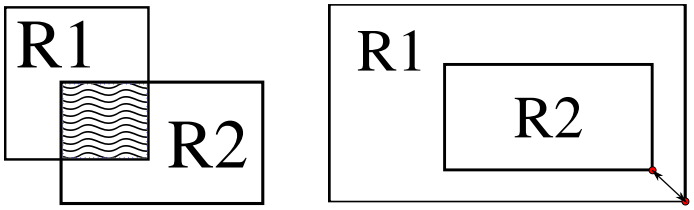
- **Vertical:** The vertical distance (left) and the difference between the horizontal centers (right).



- **VerticalStrict:** The vertical distance (left) and the difference between the horizontal coordinates of both regions (right).



- **Inside:** The area of overlap (left) and the difference between the bottom right corners (right).



3.3.3 Parsing Output

Once the parsing is performed, the structure of the mathematical expression can be obtained by traversing the derivation tree that covers the expression with the higher probability. This parsing could be presented in different ways, as a tree, MathML or LaTeX, for example. In this work the LaTeX output was chosen, but it would be easy to get other kind of output.

In Section 3.1 the mathematical expressions grammar was defined. There is an extra field that is used to obtain the LaTeX output. The process that produces the desired format output from the most probable derivation tree is just a recursive procedure from the root to the leafs. When a node is achieved, the string defined at the grammar is printed handling the child indicator ($1 or $2 , left and right child respectively). Finally, when a leaf node is reached the LaTeX representation of the symbol is printed as defined in the terminal productions files.

When the parsing process is not successful the mathematical expression is not fully recognized. But it is interesting to provide and output in spite of the expression is not absolutely correct (see Figure 3.7). In that case, the system looks for the most probable subproblem of greater size that covers the starting symbol of the grammar. In the experiments we describe how this output was used to evaluate the recognition system.

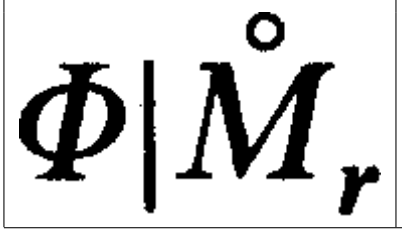
Input	Output
	

Figure 3.7: Example of partial output of the system.

3.4 Example

In order to illustrate the algorithm, we present an example using the simple grammar defined in Section 1.5 that models addition and subtraction of integer fractions.

Given the input expression

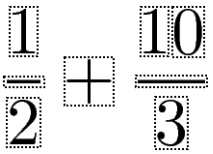
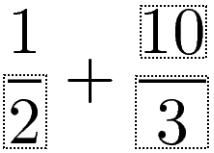
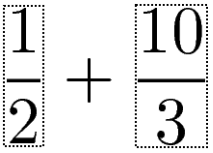
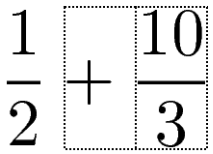
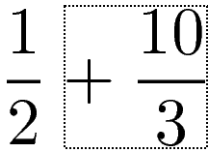
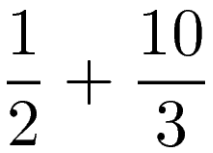
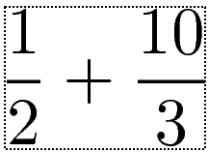
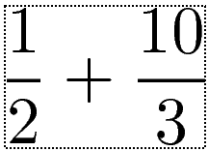
$$\frac{1}{2} + \frac{10}{3}$$

the segmentation step obtains 8 regions, which enumerated from left to right, and top to bottom are: 1, 1, 0, +, \frac, \frac, 2 3. Using this order to identify the regions and with the mathematical symbol classifier results, the level of size one of the table is initialized. Each region is labeled with its nonterminal and label that is decided by the symbol recognition module and this information is added to $t[1]$ of the parsing table. After that, the multiple connected component detection module adds to subproblems of size two the region $R_{23} = R_2 \oplus R_3$. This region of size two contains the 10 subexpression as a number 8

$$t[2] := \{(Num(8), R_{23}, 0.3952\}$$

although it is not a good choice.

Once the table is initialized, the recursion begins until it completes subproblems of size 8 (all the input regions). The following table shows the hypothesis explored by the algorithm and how the grammar models this type of expressions.

	$t[1] := \{(Num(1), R_1, 0.92),\ (Num(1), R_2, 0.89),$ $(Num(0), R_3, 0.94),(Op(+), R_4, 0.97),$ $(Over(hline), R_5, 0.81),(Over(hline), R_6, 0.90),$ $(Num(2), R_7, 0.84),(Num(3), R_8, 0.87)\}$
	$t[2] := \{\sout{(Num(8), R_{23}, 0.39)^1},(Num(10), R_{23}, 0.76)^2,$ $(OvNum(\overline{2}), R_{57}, 0.62),\quad (OvNum(\overline{3}), R_{68}, 0.71)\}$ *Note: Element^1 is replaced by Element^2 due to $\uplus$ operation*
	$t[3] := \{(Exp(\frac{1}{2}), R_{157} = R_1 \oplus R_{57}, 0.53),$ $\sout{(Exp(\frac{1}{3}), R_{268} = R_2 \oplus R_{68}, 0.37)^1},$ $(Exp(\frac{0}{3}), R_{368} = R_3 \oplus R_{68}, 0.40)^2\}$ *Note: Element^1 is replaced by Element^2 due to $\uplus$ operation*
	$t[4] := \{(Exp(\frac{10}{3}), R_{2368} = R_{23} \oplus R_{68}, 0.48),$ $(OpExp(+\frac{0}{3}), R_{3468} = R_4 \oplus R_{368}, 0.35)\}$
	$t[5] := \{(OpExp(+\frac{10}{3}), R_{23468} = R_4 \oplus R_{2368}, 0.41)\}$
	$t[6] := \emptyset$
	$t[7] := \{(Exp(\frac{1}{2}+\frac{0}{3}), R_{1345678} = R_{157} \oplus R_{3468}, 0.16)\}$
	$t[8] := \{(Exp(\frac{1}{2} + \frac{10}{3}), R_{157} \oplus R_{23468}, 0.19)\}$

Finally, the most probable hypothesis that parses the complete expression is the element

$$(Exp(\,\frac{1}{2} + \frac{10}{3}\,),\ R_{157} \oplus R_{23468}\,, 0.19)$$

Chapter 4

Experiments

Now we describe the experiments that we carried out to test the developed system. First, we describe the datasets that we used. Then, we describe the comprehensive symbol classification experiments that were done. Preliminary experiments on mathematical expressions are described at the end of this section.

4.1 Corpora

Performance evaluation of mathematical recognition systems often involves providing test data and comparing the algorithm output with the expected output. Furthermore, many classifiers needs groundtruthed data in order to train the models. There is a lack of standard datasets for mathematical expression recognition and many authors define their own datasets. Using standard datasets allows the experiments to be reproduced, to provide comparable results and it avoids the creation of corpus tuned to a particular system. For that reason, standard datasets are needed.

4.1.1 UW-III

The UW-III database [21] is a set of document images from different fields that includes 25 journal document pages containing mathematical formulae. Some of the images come from blurred photocopies. Each image has annotated the zones where the mathematical expressions are located, but the symbols are not isolated. The zones that are annotated are not embedded in the text. For this work, we isolated and classified the mathematical symbols manually[1] in order to have them available for the symbol recognition problem. The complete database had $2,233$ symbols. From this set, we removed touching symbols and those symbols that appeared less than four times. In this way the total number of symbols was $2,076$.

[1] Available at `http://www.dsic.upv.es/~jandreu/UW-III-MS.tgz`

4.1.2 INFTY

The InftyCDB-1 database [27] is a set of document images that comprises articles on pure mathematics. The database can be used both for mathematical symbol recognition and for mathematical expression recognition. Each symbol is manually annotated with its bounding box and with its class tag. Furthermore each one contains some information related to the mathematical expression it belongs to, like the expression coordinates or its LaTeX representation. Thereby, using this information it is easy to retrieve the desired information. The INFTY dataset has 21K mathematical expressions, which in turn contains 157K mathematical symbols of 212 classes. This information is annotated from 476 pages of text. Given the large size of this database, for the mathematical symbol classification task, we limited the maximum amount of training and test data. We composed four training sets of increasing size (5K, 10K, 20K, 50K) and one test set (5K). These training and test data sets were chosen at random but keeping the actual distribution of symbols of the original data set. The total number of classes for the experiments was 183.

4.2 Mathematical Symbols Recognition

The classification techniques that have been described in Section 2.2 were tested with the data sets that have been presented in section 4.1. For the k-NN classification technique, we used several values of k. We did not used any technique of prototype removing. We used the Euclidean distance between two images taking into account the difference between each pixel. We divided initially the set of prototypes into three classes based on the aspect ratio, and in this way the number of comparisons was reduced to a large extent. If we did not divide the set of prototypes according to the aspect ratio, the classification error rate remained the same. Then, the mathematical symbol images in each one of these three classes were adequately normalized. Each new prototype to be classified was just compared with the prototypes of one of these three classes depending on the aspect ratio.

For the HMM classification technique, we tested different number of Gaussian distributions per state, but the best results were obtained with 8 Gaussian distributions. Therefore, we only report results for this number of Gaussian distributions. We used left-to-right models with different number of states in each model, depending on the average width of the symbols of each class. The number of states ranged from 1 (vertical bar) to 15 (trigonometric, logarithm functions or square tail). The HTK toolkit[2] was used for these experiments. If we used the same number of states for all HMM, then the classification results were clearly worst.

[2] `http://htk.eng.cam.ac.uk/`

The SVM and WNN classifiers are used as explained in 2.2 and the experiments performed are just for tuning parameters.

For the UW-III data set, we used 25% for test and 75% for training. The test set and the training set were composed at random. Given that the UW-III data set was very small, we repeated this process 100 times. Column ≥ 4 in Table 4.1 shows the obtained average error rate for this experiments. Note that we only considered the classes that had at least 4 prototypes per class.

Table 4.1: Average classification error rate for the UW-III data set.

	≥ 4	≥ 8	≥ 16
1-NN	6.34±0.08	5.44±0.08	5.05±0.07
3-NN	8.27±0.09	6.71±0.07	5.70±0.08
5-NN	9.60±0.08	7.78±0.07	6.57±0.09
WNN	5.88±0.07	5.23±0.08	5.02±0.09
SVM	4.85±0.05	4.27±0.04	4.24±0.05
HMM-8	12.19±0.08	10.24±0.08	9.42±0.10

We can see that the best results were obtained by the SVM technique. Note also that the classification error with the k-NN rule increased as k increased. The reason for this was that as k increased, there was not enough "similar" prototypes to chose in classes with few prototypes; or in other words, this technique is very sensitive to low displacements in the bounding box. We tested this hypothesis by removing the classes that had less than 8 prototypes per class (column ≥ 8 in Table 4.1), and 16 prototypes per class (column ≥ 16 in Table 4.1). Thus, we can see in column ≥ 4 that the difference between row 1-NN and row 5-NN was 3.26, while this difference was 1.52 in column ≥ 16, which confirmed our hypothesis (experiments with INFTY data set also confirmed this hypothesis). We also observed that the WNN classification technique was very competitive.

The worst results were obtained with HMM, maybe due to low amount of training data. Thus, we observed that the difference between this classification technique and the other techniques decreased as more samples were available for training.

Table 4.2 shows the results with the INFTY data set. In all cases the results clearly improved as the size of the training set increased. The best result were obtained with SVM, but WNN obtained analogous competitive results. Note also that with a large amount of prototypes (column 50K in Table 4.2), the k-NN classification rule obtained similar values for different values of k. The worst results in this experiment were also obtained by HMM.

Table 4.2: Classification error rate for the INFTY data set.

	5K	10K	20K	50K
1-NN	6.3	4.5	4.3	3.3
3-NN	7.0	5.0	4.3	3.2
5-NN	7.9	5.5	4.4	3.5
WNN	4.8	3.5	3.4	2.8
SVM	4.5	3.4	3.0	2.6
HMM-8	8.1	7.6	7.4	7.3

In the four classification techniques, approximately 50% of the errors involved overline, minus, fractional line, underline, and hyphen symbols. These symbols are equal in all cases, and they should be distinguished by structural methods.

4.3 Mathematical Expression Recognition

Given an annotated corpus of mathematical expressions, it is difficult to perform an experiment of mathematical expressions recognition and obtain quantitative results. Usually, research works on this field use small datasets and then the results are analyzed manually [23, 11] or the evaluation method is not well explained. In this work, we wanted to experiment with a large number of expressions of the INFTY corpus and we wanted to obtain quantitative results that allowed us to compare with ground truth data.

4.3.1 Evaluation

There are several metrics defined on the literature [16] but it is difficult to use a good metric to measure structural errors, specially on complex expressions. Some of the used measures are focused in specific parts of the mathematical expression, like the symbols [28, 3], operators [6] or baselines [36]. Other metrics are centered on the placement of the expression regions [36] or the time required to complete the recognition process [14]. Finally, there are some global performance metrics like computing the minimal cost to transform a tree into another one [33]. As we described in Section 3.3.3, the output of the system developed in this work is a LaTeX expression. Unfortunately, a mathematical expression can be expressed in several ways in LaTeX, and this ambiguity is a problem in order to perform a matching to the ground truth expression. In this work, we used a evaluation measure based on the images which represents a general quantitative method to explain the quality of a mathematical expression recognition comparing the

system output with the ground truth data.

When the obtained expression is represented as an image this problem can be alleviated. For that reason, the idea is not to perform a matching with the LaTeX, MathML or tree representation, but to compare the resulting images generated by these representations. A little symbol or structural error could affect over all the expression arrangement, so a direct difference is not a good solution. However, computing a warping between the images is a way to model our intuitive idea. The *dynamic time warping* (DTW) is a well-known algorithm which performs this task efficiently in one dimension, but images are 2D data. Two-dimensional DTW is discussed in [12] and several models of lower computational complexity are presented. Due to the efficiency and the good results achieved, we use the *image distortion model* (IDM) algorithm to perform the warping between images.

The IDM algorithm in Figure 4.1 allows us to compute a distance between binary images. The input to the algorithm are two images (test and reference), and two parameters w and c. The IDM distance is computed from the test image to the reference image and it is normalized by the image dimensions. There are two parameters to be tuned. First, the size of the $c \times c$ context window, and second, the warp range w. These parameters are

---

Input: Test Image A $(I \times J)$, Reference Image B $(X \times Y)$, w , c
Output: Normalized IDM(w,c) distance from A to B

Method
for $i = 1$ to I **do** {
for $j = 1$ to J **do** {

$$i' = \left\lfloor i\frac{X}{I} \right\rfloor \ , \quad j' = \left\lfloor j\frac{Y}{J} \right\rfloor \ , \quad L = \left\lfloor \frac{c}{2} \right\rfloor$$

$$S_1 = x \in \{1, \ldots, X\} \cap \{i' - w, \ldots, i' + w\}$$

$$S_2 = y \in \{1, \ldots, Y\} \cap \{j' - w, \ldots, j' + w\}$$

$$s = s + \min_{\substack{x \in S_1 \\ y \in S_2}} \sum_{m=-L}^{L} \sum_{n=-L}^{L} ||A(i+n, j+m) - B(x+n, y+m)||$$

}
}
return $s/(I \cdot J \cdot c^2)$
End method

Figure 4.1: IDM(w,c) algorithm for binary images.

related with the resolution of the images obtained. Now we explain how these two parameters were manually tuned for this work.

The L2P[3] software was used to generate a PNG image from a LaTeX math expression. This tool has a parameter to select the resolution in *dpi*. Once the resolution was fixed, some synthetic expressions were prepared with increasing structural errors. Given a expression, common structural errors were manually added to make them representative. These errors were spatial relations misrecognition, or multiple connected components wrong interpretations. In this way, the IDM(w,c) was computed for several w, c values and as a result some plots were generated. Finally, the IDM(4,9) was selected as the metric to evaluate the performance of the classifier over the INFTY test set, because plots showed a good performance. Figure 4.2 shows the behavior of this measure. The average over some synthetic expressions showed that the more errors there are, the more distance is obtained (which was the desired behavior). We leave for future work a comprehensive study of these parameters.

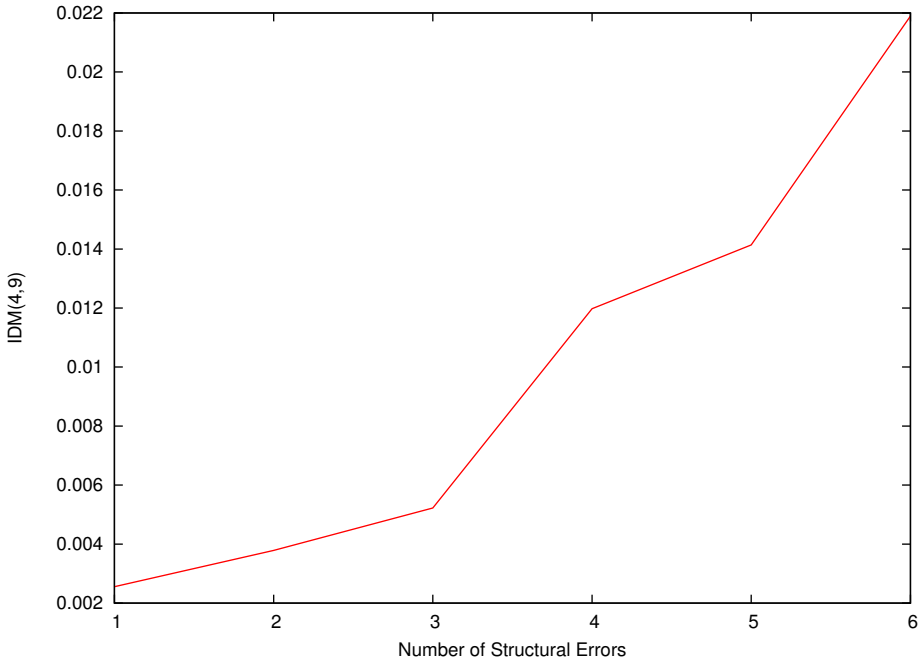


Figure 4.2: Behavior of the IDM(4,9) measure for increasing structural errors (average over 3 synthetic expressions).

4.3.2 INFTY Experiment

The INFTY corpus contains 21K annotated mathematical expressions where each symbol is described with many useful information. In order to perform

[3] http://redsymbol.net/software/l2p/

an experiment, we defined a training set and a test set. First of all, due to the large size of this dataset, we just selected those mathematical expressions that had at most a given number of symbols in the expression. The percentage of expressions in the corpus which only contained one mathematical symbol was 25.03% of the expressions, and these expressions did not contain any structural information, so they were discarded. Furthermore, the largest expression of the dataset had 108 symbols. We chose only the expressions with less than 33, which in fact represented 95% of the expressions in the database with more than one symbol (15K mathematical expressions approximately). Each symbol had several information attributes, and one of them was the *quality* of the representation, which could be: normal, touched, separate or touch_and_sep. In addition, as explained in Section 3.2.2, symbols with more than one connected component were not properly recognized, like $\leqq$ or $\cong$. For that reason, expressions that contained this kind of symbols were discarded. Finally, $3,000$ expressions were randomly selected as a test set. The train set was formed by more than 14K remaining expressions. This final step was done guaranteeing that all the symbols of the test set appeared in the training set.

As explained in Section 3, the mathematical symbol classifier used was the Nearest Neighbor (NN), due to its simplicity and performance. Moreover, the grammars used in the CYK algorithm did not need training samples, so the defined training set was only used to extract the mathematical symbols that were used as prototypes of the NN classifier.

Finally, the experiment was carried out once the system was ready to recognize mathematical expressions and an annotated test set was defined. Some results are presented using the proposed evaluation method.

The IDM(4,9) measure is 0.0 when the expression is successfully recognized. The number of expressions for which we obtained a perfect recognition result was 23.23%.

Furthermore, the developed system indicates when a recognition output does not covers all the symbols of the expression (partial output). Only 3.8% of test expressions were not fully recognized.

Ultimately, some statistics from the whole test set were extracted. The mean of the IDM(4,9) measure obtained over the $3,000$ test samples was 0.033532 and the standard deviation was equal to 0.027704. Finally, Figure 4.3 shows the IDM(4,9) value of the test set.

Although the percentage of perfect recognitions was 23.23%, some of other test samples were also successfully recognized, but there were light representation differences due to the LaTeX annotation freedom. For example, several expressions of the INFTY corpus were annotated using italic font, so the IDM distance was greater than zero in these cases (example: abc , abc). Other possible representation difference was the placement of some type of subscripts, as in the expressions $\sum_x$ and $\sum\limits_x$. We made some decisions in

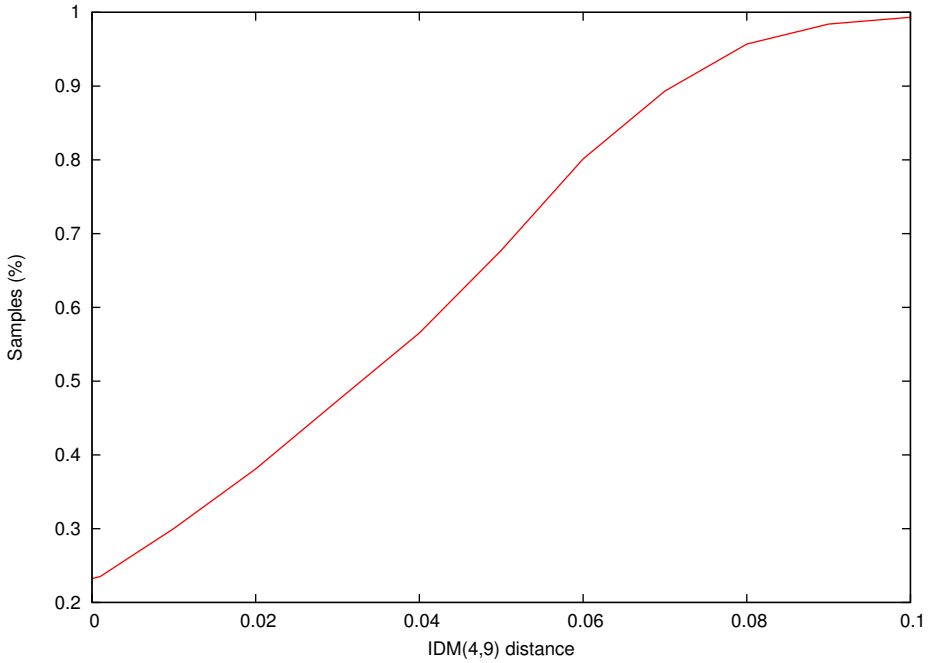


Figure 4.3: Percentage of samples with IDM(4,9) measure less than or equal to a given threshold.

order to avoid this type of differences, but in the test set there were some expressions with this evaluation ambiguity.

To our knowledge, this is the first time that the IDM measure is used to evaluate a mathematical expression recognition system. For that reason, there are some behaviors that should be studied, as is the case of the influence of the size of the expression in this distance. On the one hand, a small expression with only two symbols and one error gets a high IDM value, because half of the image is misrecognize. On the other hand, a large mathematical expression with many symbols and several errors gets a low IDM value because these errors are only a small part of the image.

The system developed is the beginning of a future PhD study about recognizing mathematical expressions using SCFG, so these IDM results acts as a baseline for future improvements. Some examples of expressions successfully recognized are shown in Figure 4.4. One of the problems of the recognizer is that the bounding box coordinates are not enough to correctly model the spatial relations among regions. For example, the expression

$$\omega = y + \sum_{|\alpha|>0} B_\alpha(y) x_1^{\alpha_1} x_2^{\alpha_2}$$

is wrongly recognized as

$$\omega = y^{+} \sum_{|\alpha|>0} B_{\alpha}(y) x_1^{\alpha} 1_{x_2}{}^{\alpha^2}$$

due to the subscript/superscript problem recognition. Other problem is the recognition of multiple connected component symbols. This can be seen in the expression

$$\mathbf{tr}(M_{ii}) = 0\{+\frac{1-\lambda}{2}\} \bmod 2$$

which is recognized as

$$tr(M_{\overline{1}i}) = 0\{+\frac{1-\lambda}{2}\} mod 2$$

where one of the i symbols is not properly recognized. In the next chapter this problems will be discussed.

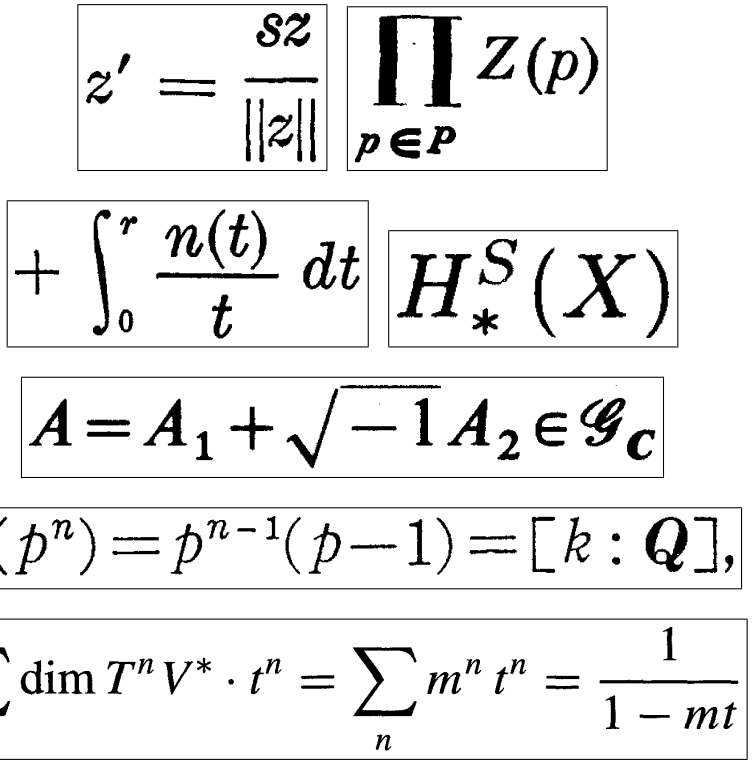


Figure 4.4: Example of expressions successfully recognized.

Chapter 5

Future Work

The system presented in this work is the beginning of a research about recognition of mathematical expressions using SCFG. Although a considerable work has been carried out, there a lot of issues to study. It should be interesting to try other symbol classifiers in the system, and many major objectives are proposed below.

5.1 Grammar Learning

In this work, the formalism used to model 2D stochastic context-free grammars only considers posterior probabilities in the production rules. However, in SCFG usually every rule has a fixed probability. A very interesting objective is to use the *inside-outside algorithm* to learn the probabilities of the production rules of the SCFG from a annotated corpus.

5.2 Multiple Connected Components and Noise

The developed system represents a starting point for a future research, and some issues deserve future comprehensive research. Major problems are to deal with noise, and to deal with multiple connected components recognition.

Currently, the system assumes that the images do not present (to much) noise, and an image filter is applied before a mathematical expression is provided to the recognizer. The multiple connected components detection method is explained in Section 3.2.2, and it works fine recognizing mathematical symbols which are split in two connected components. But it is not enough to correctly recognize many cases. For example, one symbol could be split into 3, 4 or 5 connected components due to image degradation. Therefore, a better way to deal with these problems must be researched.

5.3 Spatial Relations

In Section 3.3.2 are explained the probability distributions used to model the spatial relations among regions of a mathematical expressions. But these models are very simple. It is necessary to employ more features that could help to obtain better models. For example, the baseline information of a region should contribute substantially in the performance of the system. In summary, the bounding box coordinates aren't enough to correctly determine the spatial relations between regions, so more features are needed.

The spatial distributions used are manually defined. So, despite adding new features, it should be very interesting to be able to automatically learn the parameters of these distributions.

5.4 Reducing Complexity

Currently, the cost of parsing one mathematical expression is $O(|P|n^3 \log n)$. When the number of symbols of the expression is large, this complexity could be very expensive. For that reason, it is necessary to reduce this computational cost.

On the one hand, using thresholds to discard hypothesis improves the computational time of the algorithm, but it could deteriorate the quality of the parsing process. However, it would be very interesting to employ A^* parsing techniques because it improves the time cost of the system keeping the parsing quality.

On the other hand, it would be interesting to try algorithmic improvements. It could be possible to improve the working of the data structures used, specially that involved in Section 3.3.1. Furthermore, techniques like divide and conquer could be used to split a mathematical expression and then perform parallel parsings of lower dimension.

5.5 Handwritten and On-line

The presented work tackles the recognition problem of off-line printed mathematical expressions, but the techniques, models and algorithms used can be adapted to deal with other modalities. The idea is to be able to recognize handwritten and on-line expressions, too. These cases have several differences with the present work and so, many other problems that should be considered. The HMM mathematical symbol recognition is a new method used that could be more interesting to classify handwritten mathematical symbols.

Bibliography

[1] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[2] Francisco Álvaro and Joan Andreu Sánchez. Comparing several techniques for offline recognition of printed mathematical symbols. In *Proceedings of 20th International Conference on Pattern Recognition*, Istambul, Turkey, 2010.

[3] Kazuki Ashida, Masayuki Okamoto, Hiroki Imai, and Tsubasa Nakatsuka. Performance evaluation of a mathematical formula recognition system with a large scale of printed formula images. *Document Image Analysis for Libraries, International Workshop on*, 0:320–331, 2006.

[4] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. A hybrid classifier for handwritten mathematical expression recognition. In Laurence Likforman-Sulem and Gady Agam, editors, *DRR*, volume 7534 of *SPIE Proceedings*, pages 1–10. SPIE, 2010.

[5] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: A survey. *International Journal on Document Analysis and Recognition*, 3:3–15, 1999.

[6] Kam-Fai Chan and Dit-Yan Yeung. Error detection, error correction and performance evaluation in on-line mathematical expression recognition. *Pattern Recognition*, 34(8):1671 – 1684, 2001.

[7] P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Proceedings of the SPIE Visual Communications and Image Processing IV*, pages 852–863, 1989.

[8] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, 2002.

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

[10] U. Garain, B.B. Chaudhuri, and R.P. Ghosh. A multiple-classifier system for recognition of printed mathematical symbols. In *Proc. ICPR*, volume 1, pages 380–383, 2004.

[11] B. Q. Huang and M-T. Kechadi. A structural analysis approach for online handwritten mathematical expressions. *International Journal of Computer Science and Network Security*, 7(8), 2007.

[12] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Ney. Deformation models for image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1422–1435, 2007.

[13] Andreas Kosmala and Gerhard Rigoll. On-line handwritten formula recognition using statistical methods. In *In Proceedings of the 14th International Conference on Pattern Recognition*, pages 1306–1308, 1998.

[14] Andreas Kosmala, Gerhard Rigoll, Stephane Lavirotte, and Loic Pottier. On-line handwritten formula recognition using hidden markov models and context dependent graph grammars. *Document Analysis and Recognition, International Conference on*, 0:107, 1999.

[15] Leslie Lamport. *Latex: a document preparation system.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[16] Adrien Lapointe. Issues in performance evaluation of mathematical notation recognition systems. Master's thesis, Queen's University, Kingston, Ontario, Canada, 2008.

[17] Seong-Whan Lee and Dae-Seok Ryu. Parameter-free geometric document layout analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1240–1256, 2001.

[18] Christopher Malon, Seiichi Uchida, and Masakazu Suzuki. Mathematical symbol recognition with support vector machines. *Pattern Recogn. Lett.*, 29(9):1326–1332, 2008.

[19] Akihiro Nomura, Kazuyuki Michishita, Seiichi Uchida, and Masakazu Suzuki. Detection and segmentation of touching characters in mathematical expressions. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 126, Washington, DC, USA, 2003. IEEE Computer Society.

[20] R. Paredes and E. Vidal. Learning weighted metrics to minimize nearest-neighbor classification error. *IEEE Transaction on Pattern Analisys and Machine Intelligence*, 28(7), 2006.

[21] I. Phillips. Methodologies for using UW databases for OCR and image understanding systems. In *Proc. SPIE, Document Recognition V*, volume 3305, pages 112–127, 1998.

[22] Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:63–84, 2000.

[23] Daniel Průša and Václav Hlaváč. Mathematical formulae recognition using 2d grammars. *International Conference on Document Analysis and Recognition*, 2:849–853, 2007.

[24] T.H. Rhee and J.H. Kim. Efficient search strategy in structural analysis for handwritten mathematical expression recognition. *Pattern Recognition*, 42(12):3192–3201,, December 2009.

[25] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[26] Faisal Shafait, Joost van Beusekom, Daniel Keysers, and Thomas M. Breuel. Structural mixtures for statistical layout analysis. In *Document Analysis Systems*, pages 415–422, 2008.

[27] M. Suzuki, S. Uchida, and A. Nomura. A ground-truthed mathematical character and symbol image database. In *Proc. 8th International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 675–679, 2005.

[28] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. Infty - an integrated ocr system for mathematical documents. In *Proceedings of ACM Symposium on Document Engineering 2003*, pages 95–104. ACM Press, 2003.

[29] Ernesto Tapia and Raul Rojas. Recognition of on-line handwritten mathematical formulas in the e-chalk system. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR*, pages 980–984, 2003.

[30] Xue-Dong Tian, Hai-Yan Li, Xin-Fu Li, and Li-Ping Zhang. Research on symbol recognition for mathematical expressions. In *ICICIC '06: Proceedings of the First International Conference on Innovative Computing, Information and Control*, pages 357–360, Washington, DC, USA, 2006. IEEE Computer Society.

[31] A.H. Toselli, A. Juan, and E. Vidal. Spontaneous handwriting recognition and classification. In *Proc. of the 17th International Conference on Pattern Recognition*, pages 433–436, Cambridge, UK, August 2004.

[32] Kenichi Toyozumi, Naoya Yamada, Kenji Mase, Takayuki Kitasaka, Kensaku Mori, Yasuhito Suenaga, and Tomoichi Takahashi. A study

of symbol segmentation method for handwritten mathematical formula recognition using mathematical structure information. *Pattern Recognition, International Conference on*, 2:630–633, 2004.

[33] Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.

[34] Ba-Quy Vuong, Siu-Cheung Hui, and Yulan He. Progressive structural analysis for dynamic recognition of on-line handwritten mathematical expressions. *Pattern Recogn. Lett.*, 29(5):647–655, 2008.

[35] Ryo Yamamoto, Shinji Sako, Takuya Nishimoto, and Shigeki Sagayama. On-line recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar. *IEIC Technical Report (Institute of Electronics, Information and Communication Engineers)*, 2006.

[36] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(11):1455–1467, 2002.