Mathematical Formula Recognition using Machine Learning Techniques

Théodore Bluche
under the Supervision of Dr. Vasile Palade

Dissertation submitted for the completion of the
MSc in Computer Science

Worcester College
University of Oxford

September 3, 2010

Abstract

Unlike texts, printed mathematical expressions have a two-dimensional nature, and their recognition involves the recognition of the structure of the formula and of the symbols contained in it. In most research, the structure recognition uses the identity of symbols. Moreover, a few previous works use the structure in the symbol recognition, or machine learning techniques in the structure recognition part.

There is a lot of mathematical symbols, they can be very similar, and new symbols are being invented by scientists. The arrangements of symbols, however, evolve less. In this dissertation, we investigate the possibility of recognizing the structure of the mathematical expression without the symbols' identity. We also perform a prior classification of the symbols using the structure only. Moreover, we use very few knowledge about mathematical expression syntax. This approach has, to the best of our knowledge, never been tried.

We built a system using machine learning techniques, such as neural networks and fuzzy logic. A binary image representing a mathematical expression is segmented, and the recognition is performed using the symbols' bounding boxes. An iterative algorithm using a multi-classifier system recognizes the structure and classifies the symbols.

The results proved that the context of a symbol, when known and used, can help classify the symbol. The structure recognition using a non-recursive algorithm with very few backtracking yielded good results. This project proved that the symbols' identity is not essential for the structure analysis. Moreover, the structure recognition provides some useful information for the symbol classification. This classification is assumed to help the symbol recognition in a future work. Finally, the machine learning approach produced a flexible system, able to adapt to unknown symbols and writing styles, and to return confidence values for the recognition, rather than a crisp interpretation.

Acknowledgements

I owe my deepest gratitude to my supervisor, Vasile Palade, for the guidance and support he gave me thoughout this project. His advice and experience have greatly helped me focus on the important points and write this dissertation.

It is an honor to thank Dr. Dorothea Blostein, who published several papers on the subject, and who pointed me out the Infty data set. I owe a special acknowledgement to Dr. Claudie Faure, researcher in ENST ParisTech. She made available to me two papers she published, which I could not find. I am very grateful for she showed great interest in my project, and replied all my emails quickly. I would also like to thank Laurence Likforman, who interviewed me during a PhD application, and then sent me an interesting paper related to my project [18].

This thesis could not have been complete without the help of the persons who have completed the online quiz for manual equations labelling. I collected more answers than expected, and some people even sent me suggestions about the presentation of the quiz.

I would like to show my gratitude to my academic supervisor, Bernard Sufrin. I enjoyed our conversations, and his support has been of immeasurable value. His encouragement and guidance thoughout the masters allowed me to take pleasure in studying in Oxford.

I also want to take this opportunity to thank the teachers, staff and students of the Computing Laboratory, and of Worcester College, for their assistance and friendship, and without whom the experience of being a student in Oxford would not have been complete.

I owe an acknowledgement to Supélec, the engineering school in which I studied last year. They helped me applying to Oxford University. I would like to address a special thanks to Anne Chrétien, for her recommendations and support, and my referees Yolaine Bourda, Alain Destrez and Françoise Brouaye. I would like to thank my uncle Olivier, who shared with me his experience living in England and helped me applying to Oxford University

Finally, I am indebted to my family for they made this fulfiling experience possible by funding my MSc. They have been of great support when I encountered difficulties, from the application process to the end of the project.

Contents

Chapter 1

Introduction

First, we will briefly describe the problem of mathematical formula recognition. Then, we will state the aims of the project, and, finally, we will introduce the different parts of this document.

1.1 Description of the Problem

Today, as documents tend to be dematerialized to be stored on computer, systems able to transform a physical item into a digital one are needed. Most documents mainly contain text, either printed or handwritten. To this concern, Optical Character Recognition (OCR) has had tremendous improvements since the 1950s, leading to efficient pieces of software available nowadays. In particular, current technology allows one to enter a text by directly writing it on a data tablet, or touchscreen.

While some may argue that typing a text is quicker than writing it, there is no doubt that it would be easier to write or scan a mathematical expression rather than inputting it using the available tools. For example, writing a complicated equation using Latex requires expertise, while the equation editors such as the one available in MS Word involve the selection of a symbol or a structure in a list, which constitutes a long process. Moreover, several printed documents, such as scientific papers or books, cannot be efficiently stored on a computer unless a system is able to recognize the mathematical formulae they contain.

Plain texts and mathematical expressions are very different items. Text is a one-dimensional sequence of characters, whereas mathematical expressions are a two-dimensional arrangement of symbols. Text contains letters and digits, whereas the number of symbols in equations is infinite. Moreover, in texts, single characters do not have a meaning of their own (except for acronyms). The smallest meaningful entity is the word, which is a sequence of characters. In mathematical expressions, each symbol has its meaning. In fact, mathematical expressions provide a convenient way to communicate scientific concepts in a short and clear way. Indeed, $\sum_i a^i$ looks somewhat simpler than "*the sum of all powers of the number a*". This example also illustrate the fuzziness surrounding symbols. If a and b refer to the same concept, $\sum_i a^i$ and $\sum_j b^j$ have exactly the same meaning. The set of expressions with that meaning is infinite, whereas in texts, characters are chosen to form words which are part of a vocabulary. Synonyms may exist but (i) in a finite number and (ii) they do not express the exact same idea.

The symbolic style of mathematical expressions drives scientific writers to explain what concept the symbols refer to. Indeed, the same symbol can have different meanings. For example, the capital Greek letter sigma (Σ) often refers to the summation, but might represent a set of object in a different context. Letters are sometimes variables which need to be described to be understood.

These differences between text and mathematical expressions imply that their recognitions are dissimilar. In texts, characters can be recognized in a sequence and grouped to form words. A vocabulary can for instance help disambiguate the recognition. For mathematical expressions, the symbols must be recognized and the structure as well. Disambiguation is more complex because there is no such thing as a mathematical vocabulary.

Although texts and mathematical expressions differ in their form, similar purposes exist for their recognition. This often corresponds to what extend one wants to understand the input. This includes:

- the mere recognition of characters/symbols and their position to be able to reproduce the input on a computer,
- the recognition of whole words and formulae to ensure a meaningful representation of the input,
- the understanding of the input, for example using natural language processing for texts. For mathematical expressions, it corresponds to the understanding of the meaning of the symbols.

The first published works on mathematical expression recognition date from the 1960s, but this field enjoyed significant research interest in the last two decades. The symbol recognition is either performed by classical OCR techniques, for instance using support vector machines [22], pattern matching [16, 24], or taken as granted (e.g. [28, 20, 2, 1, 9]). The structure analysis is mainly done using geometrical considerations, based on implicit rules (e.g. in [21, 28, 20, 27]) or grammar rules [17, 15, 8]. These rules take into account the identity of the symbol, and very few papers, such as [25], attempted to dissociate the structure analysis from the symbol recognition. The ambiguity in mathematical expressions, especially when handwritten, is commonly accepted. It can be an ambiguity on the symbol identity (e.g. q and 9 can look similar) or on the structure. Only some researches dealt with that problem, by keeping several interpretations on the symbol's identity until the structure disambiguate it [17], or by producing several final interpretations, for the user to choose the best one [29]. Even though artifical intelligence can be found in the structure recognition, through fuzzy logic [29, 12], or search algorithms [9, 17, 18] (among others), machine learning is, to the best of our knowledge, rarely used in the structure analysis [1, 9].

1.2 Project Aims

The structure analysis of mathematical expressions varies from the mere recognition of spatial relationships (on the same line, exponent, and so on) to the real understanding of a formula. For example, Zanibbi et al. [28] perform a lexical and semantic analysis of the formula, enabling their system to recognize the function *sinus* rather than a triplet of symbols ('s', 'i', 'n') on the same line. The semantic analysis is often used to recognize the

structure. For instance, some papers, such as [27], utilize the dominance and precedence of symbols to find the most likely relationships.

Most papers use the identity of symbols to recognize the structure. For example, it allows to deduce the dominance of the symbol, and the regions where arguments, such as exponent, should be found. It also means that an error on the symbol recognition can have severe consequences on the structure analysis. In particular, mathematical notation evolve with the needs of scientists, and new symbols appear. These symbols might not be identified and compromise the structural analysis.

This project investigates *the possibility of performing the structural analysis and a symbol classification at the same time*, using only very few knowledge about mathematical expression syntax. Our approach is based on the mutual constraint existing between symbols and structure. Indeed, knowing the symbols' identity helps analyse the structure, but the structure can help disambiguate the symbol recognition ([17]). We aim to develop a system based on machine learning techniques, where the symbol classification and the structure analysis are separate tasks which constrain each other.

Instead of identifying the symbols, we classify them according to a classification described in the literature (e.g. [28, 17]). *Our contribution to the field consists in assuming that this classification can be performed using the symbols' bounding boxes, and the structure of the expression only.* We develop an iterative algorithm to exploit the constraints between both tasks. *Our motivation is to define a method for building a system as flexible as possible and able to deal with the evolution of mathematical notation*, and with the variation in the writing style. To do so, we implement machine learning artifacts, which allow us to return a soft interpretation, with confidence values, rather than a crisp one. The goal is to dissociate the structure recognition, which is the core of a mathematical expression recognizer, from the actual symbol identification, while providing results to improve the latter. Indeed, the classification of symbols constrains their identification, making it easier.

Last but not least, we aim to achieve a very fast recognition of the structure. This implies a limited usage of backtracking, avoiding too much recursion, and a few comparisons between symbols only. This is a big challenge, given the nested two-dimensional nature of mathematical expressions.

1.3 Structure of the Thesis

This thesis is divided into seven chapters.

- Chapter 1 introduces the project and its relation to the field of mathematical expression recognition.
- Chapter 2 presents a background study about mathematical expressions in general, and their recognition by a computer program in particular. This is followed by a review of the literature in the area.
- Chapter 3 explains our motivations and hypothesis for this new approach to mathematical formula recognition, and gives an overview of the method we proposed.

- Chapter 4 provides more details about the design of the system, that is the artifacts and parameters used, and presents the algorithms.
- Chapter 5 focuses on the actual implementation of the system using object-oriented programming
- Chapter 6 evaluates the results yielded by the implemented system and draws conclusions about the performance.
- Chapter 7 concludes the dissertation, analysing how this approach can contribute to the field of recognition of mathematical expression, and giving ideas for further development.
- Finally, the appendices provide more details about the developed system and its performance.

Chapter 2

Background

In order to build a system able to recognize mathematical expressions, some preliminary work is necessary. We need to specify what is a mathematical expression. That means that we have to define how mathematical expressions are written, and how we read them. We have to investigate the different existing arrangements of symbols, and the corresponding meanings. Moreover, mathematical expressions can be used in different ways. They may be written for humans to easily read them. They are then put down as two-dimensional graphics. They may be written for computers to use it. The representation can vary, from the reverse polish notation used in 1980s handheld calculators, to tree structures in some symbolic computation systems today.

For mathematical expression recognition, the notation which we want to understand is graphical. It poses some problem, such as understanding the relations between symbols. Indeed, they have a structure, used for mathematical communication between humans, independently from computers. This leads to the identification of the tasks involved in expression recognition, each having specific problems.

Section 2.1 focuses on the mathematical notation and representation of the formulae in general. Section 2.2 explains what is mathematical expression recognition. Some research has been carried out in the past, especially in the last few years. Section 2.3 aims to review the state of the art in the field of mathematical formula recognition.

2.1 Mathematical Notations

A mathematical expression is not merely symbols put down in a random two-dimensional layout. It has a well-organized structure, which obeys the rules of the mathematical notation. The arrangement of two symbols with respect to each other conveys a certain meaning.

2.1.1 Writing and Reading a Mathematical Formula

The usual writing and reading order for mathematical expressions is left to right. Therefore, understanding a mathematical expression is not completely two-dimensional. Moreover, symbols in a relation are usually close to each other. For example, a subscript is very likely to be the closest symbol on the bottom right of its parent. In a few cases, the interpretation of a formula is possible only by reading the whole formula. For instance, we can fully understand an integration when we see the final dx.

However, reading a mathematical expression is not straightforward, for different symbols are usually read differently. This can be illustrated with some examples. $\sum_{i=0}^{n} a^i$ is usually read "sum over i from 0 to n of a to the ith". The pattern is *operator, argument below, argument above, argument on the same line.* $\frac{a}{b}$ is read "a over b". The pattern in this case is *argument above, operator, argument under.*

2.1.2 Problems posed by Mathematical Expressions

The major differences between plain text and mathematical expressions are: (i) there are far more symbols used in mathematical formulae, and (ii) there are more kinds of relationships in mathematical expressions. Those two points induce two main challenges in the recognition of mathematical expressions.

The Number of Symbols

Plain texts contain letters, digits and punctuation. There are some general rules such as "capital letters are found at the beginning of a word" or "digits are usually not mixed with letters in the same word". Mathematical expressions can contain a wider range of symbols. Roman letters and digits are only a small subset. The Greek alphabet is often used, and a lot of mathematical symbols exist. It is impossible to list them, since scientists keep inventing new ones. Moreover, there are symbols which are similar in shape and appear in different contexts:

- $\prod$ is the product symbol, whereas π is the Greek letter pi,
- P is the capital letter while p is the small letter,
- $<$ is the operator 'lower than' in $1 < 2$ and a bracket in $\langle a, b \rangle$
- s is a variable in s^3 whereas it is part of a function name in $sin(\frac{3\pi}{2})$

or have distinct functions or meanings :

- digits almost always represent numbers
- letters can either be included in a function name, like n in $sin(\pi)$, represent an precise object which is explained in a short text, as in "$O(n)$, *, where n is the number of nodes*", or be artifact to represent a more general concept, as the hidden variable n in $\sum_{n=0}^{5} x^n$.
- finally, most symbols are redefined for the need of the writer.

The Spatial Relationships

We can list six main spatial relationships: superscript (or top-right), subscript (or bottom-right), on the same line, above, under, inside. Once again, scientists may use new spatial arrangements when they need to, or give different meanings to existing relationships. For example, with the 'superscript' relation, i is the power in x^i, whereas (i) is an index in $x^{(i)}$. The relation 'on the same line' is even more complicated, for example:

- in $\sum a$, a is an argument,
- in 42, 4 and 2 are part of the same number,

- in an, the meaning is likely to be $a \times n$
- tan represents probably the 'tangent' function
- $(i + 1)$ can be the argument of a function, of a product, and so on.

While the meaning can differ from one case to another, the spatial behaviours associated with these relationships is fixed, and the translation in a Latex form is often the same for all these possibilities.

Mathematical can generally be considered as nested structures, especially due to the existence and nature of spatial relationships. In $\frac{a+1}{a-1}$, $a + 1$ and $a - 1$ are mathematical expressions on their own, nested in a more general one. As is $2x_k$ in n^{2x_k}. Because formulae are globally written on a line, called baseline, the nested structure implies the existence nested baselines. The nested nature of mathematical expression entails a relation of child to parent for the nested expression.

2.1.3 Representing a Formula

The most intuitive way to represent mathematical expressions is graphical. It is a printed or handwritten two-dimensional structure, with symbols of different size and positions. It gives an easy reading and understanding of the formula by humans.

However, other forms exist, and are suited for inputting expressions in a computer. Simple formulae can be written on a single line. For example, `(x+1)*y^i` remains one of the simplest representation of that expression. Understanding and interpreting it is straightforward for anyone knowing a few mathematics. It becomes harder when the complexity increases. Latex-like forms allow one to input a mathematical formula using a simple keyboard. Complex symbols are represented by keywords. For example, $\sum_{a=0}^{N} a$ becomes `\sum_{a=0}^{N} a`.

To capture the meaning of an expression in a logical manner, it can be put in the form of a tree. For example, the previous formula `(x+1)*y^i` is represented by the tree on Figure 2.1.

Another format is MathML [1], which is an XML format. It captures the nested nature of mathematical expressions. Two variants exist. Presentation MathML focuses on the spatial representation of the expression. Content MathML represents the meaning of it, and can be seen as a text translation of the tree representation presented before.

2.2 Mathematical Formula Recognition

Mathematical formula recognition is the task in which the image representing a mathematical expression is interpreted by the computer so that it can be stored, interpreted, and reused. Several purposes can be identified. The goal can be to recognize a handwritten expression, and display it in an electronic form. In this case, the meaning of the expression is not important. Sometimes, the meaning is important, when one wants to fully understand the expression. In this case, one has to recognize implicit multiplications,

[1] Mathematical Markup Language, W3C, from http://www.w3.org/TR/MathML2/, July 2010

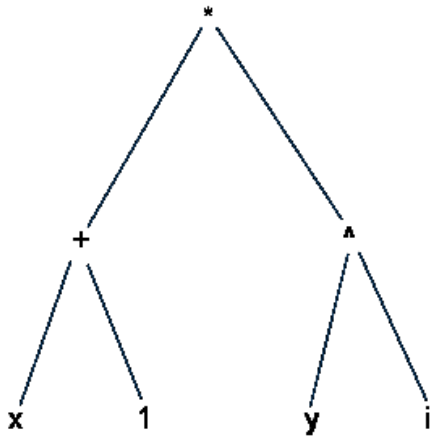

Figure 2.1: Representation of (x+1)*y ^ i as a tree

for example (e.g. ax means $a \times x$). At a higher level, it is useful to retrieve the meaning and to forget the form of the formula. For example, $f(x)$ and $g(y)$ can have the same meaning in different contexts.

The recognition of mathematical formulae consists of two tasks:

- The recognition of the symbols: each foreground pixel in the image belongs to a symbol, and each symbol has a meaning in the expression, and conveys some information.
- The recognition of the structure: the two-dimensional layout of an expression obeys some rules, and each arrangement corresponds to a particular meaning.

2.2.1 Symbol Recognition

The symbol recognition task is the procedure by which each symbol is isolated and recognized by a classifier. It is not an easy task because of the large number of symbols, and we do not have a dictionary of words as we do for text recognition. The same symbol can appear in different context, and it is sometimes crucial to make the difference between for example $\sum$, the summation symbol and Σ, the Greek letter. Some different symbols have the same shape, such as p and P, and it is no easy task for a classifier to recognize them properly. Even more issues appear when it comes to handwritten expressions (e.g. the q-9 problem pointed out by [17], see Figure 2.2).

This task is usually performed by neural networks or support vector machines. In most published papers on the topic, the symbol recognition is done (or assumed to be successful) before the structure recognition. It can be problematic, because an error on the symbol identification can have serious consequences on the structure recognition. For example (x) might be recognized as a subscript in $P(x)$ if P is identified as the small letter p. Some researches [17, 9, 18] propose to leave ambiguities on the symbol recognition until it is disambiguated by the structure.

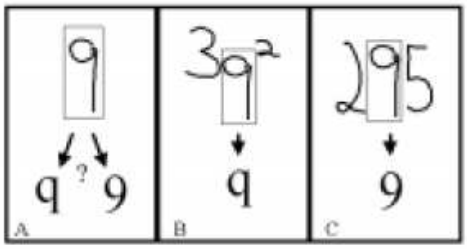

Figure 2.2: The q-9 problem exposed in [17]

2.2.2 Structure Recognition

While the previous task is easy to understand, the recognition of the structure can be more or less complex, according to which level of interpretation one wants to achieve. We have identified several purposes, along with the extend to which the structure must be analysed.

- For a mere digitalizing in a Latex form for example, the recognition of spatial relationships regardless of their meaning is usually sufficient. For instance, in the example presented earlier: x^i / $x^{(i)}$, we do not need to know that this i is a power in the former case, and an index in the later to write in Latex `x^i` and `x^{(i)}`.

- One may want to understand the sense of the expression. It can be important to know that $1 + 2$ is not only a sequence of symbols on the same line, but an addition. In this case, the meaning of the operators, both explicit (e.g. $+$) and implicit (e.g. the multiplication in xy), must be known.

- Sometimes, an even deeper analysis is required. In mathematical expressions, the meaning does not really come from the symbols involved, but rather from what they represent. As pointed out earlier, the same function (or variable) can have a different name in different contexts. It is possible that $f(x)$ and $g(y)$ are the exact same thing in two different documents.

Thus, the recognition of the structure of an expression is often a mixture of a layout analysis and the interpretation of what represent the symbols and the relationships.

2.3 Review of Existing Techniques

As we will see, some methods are only based on spatial considerations, such as baselines. Other techniques use rule-based systems such as grammars, and parse the expression to find its interpretation. Several algorithms take into account knowledge about mathematical symbols and operators, and their spatial behaviour. Very few papers use machine learning techniques, although there have been some works using fuzzy logic.

Zanibbi et al. [28] analyse the baselines present in an expression. In particular, they consider the dominant baseline, which is the line on which the expression would be written, and nested baselines, corresponding to subscripts, for example. During a first step (layout pass), they build a tree based on these baselines. Then, they use knowledge about

mathematical notation properties to define some tree transformations. For example, they recognize that the sequence '*sin*' correspond to the so-called function, or that in $\lim_{n\to 1}$, $n \to 1$ is in fact an argument of the limit. Finally, they obtain a tree which represent the meaning of the equation. The Baseline Structure Tree building requires a recognition of individual symbols. They define symbols classes, such as *ascender* (e.g. 'd', 'b'), *descender* (e.g. 'y', 'p'), *Variable Range* (e.g. $\sum$, $\bigcup$), and so on. Using these classes, they define regions around a symbol for where subscripts, superscripts, above expressions (and so on) should be found, if any. This baseline structure analysis has been used in several other works, such as [22].

In a later work [29], Zanibbi et al. improved the recognition of subscripts and superscripts, using fuzzy regions instead of crisp regions. This was motivated by the fact that most ambiguities in handwritten mathematical expressions concern the choices subscript/inline and inline/superscript. Besides, the use of fuzzy logic enables them to return a ranked list of interpretations, instead of just one.

The online recognition method proposed by Suzuki et al. [20, 24] proceed in four steps: stroke recognition, structure recognition, multiple-stroke character recognition, and subscripts/superscripts recognition. After a baseline analysis, they first recognize dominant symbols such as $\sum$ or fraction lines, which they call *parent symbols*, and identify *child blocks* in which arguments should be found. Then the rest of the structure is recognized using thresholds for the heights of bounding boxes and vertical locations of the characters.

Lee et al. [16] use a procedure-oriented algorithm. After the symbol recognition, they extract operators such as $\sum$, and group them with surrounding symbols, according to the operator spatial behaviour. For example, $\sum$ is grouped with symbols placed above and below. Then, they order these groups according to their y-coordinate. Finally, a string representing the expression is generated.

Chang [6] first identifies the operators, and build a tree representation. He uses pattern matching, with patterns built from previous knowledge about how symbols should be organized around these operators. Operator precedence is used to correctly understand the expression.

Chaudhuri et al. [7] have a more complete three-step approach. Their system includes first the detection of an ME in a document. then comes the symbol recognition. Finally, they study the arrangement of symbols. To do so, they group symbols using their spatial relationships involving features such as bounding boxes and sizes. The identification of logical relationships among different groups comes after, and is driven by a set of rules.

In [21], Tapia and Rojas first get the baselines, as in [28], and recursively build a minimum spanning tree, in which each node is a symbol. They define the distance between two symbols according to whether one lies in the range of the other (geometrically speaking) or not. In this method, they use different classes of symbols (ascender, descender, centered), and regions around a symbol are built accordingly with crisp thresholds. They also consider attractor points according to the operator class, expressing where arguments are usually found. For example, for the operator $\sum$, there usually are arguments above and below, so they put attractor points on the middle of the upper and lower boundaries of this symbol.

Xiangwei et al. in [27], also use symbol dominance and minimum spanning tree, but perform further analysis of symbol arrangements in order to build symbol groups more accurately. In particular, they identify the main character in the dominant baseline, or group symbols which represent a function name (e.g. '*sin*', '*lim*').

Ha et al. [13] use a combined strategy to analyze the structure. Besides the baseline structure analysis as proposed in [28], but using a graph to represent the expression, they construct a minimum spanning tree, as in [21]. They perform afterwards a syntactic and semantic analysis, using rules based on the operator identity. They have a special treatment for ambiguous operators such as horizontal lines.

Suzuki et al. [9] use a *virtual link network*. During the character recognition phase, they give a confidence value for each possible symbol interpretation. They build then a network in which each node corresponds to a symbol. It contains information about the symbol such as its bounding box and the possible interpretation. Edges between nodes correspond to a possible interpretation for the pair of symbols, and a possible relationship (e.g. subscript). A score is given for a possible relationship and is the cost of the corresponding edge. They consider possible minimum spanning trees, and only keep the valid ones. An invalid tree has for example an incoherent labeling. The previous steps corresponded to a local analysis. A global score is given to each tree, according to the total cost of the tree, and some penalties. The tree which have the minimum score represents the expression. It corresponds to a unique interpretation (disambiguation) of each symbol, and of each relationship.

Rhee and Kim [18] presented a method to perform an efficient search to recognize the structural analysis. They are dealing with handwritten expression. They propose a multi-layer tree, to globally disambiguate local ambiguities. At each level in the search tree, they add one stroke, and find the most likely relationships it can have with the already considered stroke. Then each level is composed of a full interpretation of the structure up to now, and when a stroke is added, several nodes are created for the possible identities of the character, and the different relationships it can have with the rest of the formula. They define a cost and an heuristic, and perform a *best-first search*. To reduce the search space, they consider symbol dominance, and more importantly, they delay the symbol's identity recognition.

Miller and Viola [17] maintain ambiguity during the character recognition step. Then, they calculate the probability for each symbol to be in a certain class (small letter, digit, binary operator, and so on), and the probability to be a subscript, superscript, or inline expression, according to the symbol recognition and to some layout properties. Then, they use a stochastic context-free grammar (SCFG) to parse the expression. The system uses convex hull instead of bounding boxes, models positions of characters with a Gaussian distribution, and uses an A-star algorithm to search the space of possible interpretations.

Chen et al. [8] perform both the recognition and the understanding of the formula. After the character recognition, they use a set of rules. There are three sets of rules used one after the other. *Mathematical rules* are grammar rules for parsing and understanding the expression. *Sense-based rules* are used to disambiguate the layout. *Experience-based rules* handle uncertainty in expression semantics. The last two rule bases are learnt and modified

by feedback. Afterwards, they use the results to build a layout tree and a semantic tree, and then convert them into a LaTeXstring.

Tian et al. [23] propose a rule-based system for both character and structure error correction. More precisely, they recognize the expression using the method proposed by Zanibbi et al. [28], based on BST. Then they apply some grammar rules to correct common errors.

Garcia and Coüasnon [11] used the grammatical formalism Enhanced Position Formalism (EPF) to describe the layout of an expression, and also the geometry of some symbols. They can use it to parse a formula and recognize symbols made of lines.

Lavirotte and Pottier [15] focus on the syntactical analysis and use context-sensitive grammar rules. They build a graph with information about the characters and their relative size, and use graph-rewriting rules based on a context-sensitive grammar, which allows them to perform disambiguation at the same time.

Awal et al. [3] try to optimize the symbol segmentation and recognition, and the structure recognition at the same time, for handwritten expressions. They group strokes together, and calculate a cost function based on a *recognition score* for this group to represent a symbol and a *structural score*, taking into account the height and baseline of the symbol, and some knowledge about it. The symbol recognition and segmentation uses a neural network, and the structure of the expression is recognized using grammar rules.

Wang and Faure [25], have another approach to this problem. They do not use any information concerning the identity of the symbol. According to the relative height of two symbols, they define three situations. For each of them, they build a probability distribution for the relation (subscript/inline/superscript) between symbols, according to their relative vertical location. Moreover, they take into account the horizontal distance between the symbols. Indeed, a vertical offset of a far symbol is not very likely to mean that this symbol is a superscript rather than inline, but more probably due to the irregularity of handwriting. They consider symbols three by three, to eliminate some ambiguities. For example, with this technique, the symbol c in the expression $a^b c$ will not be considered as being a subscript of b. They add some constraints on possible triplets relations and a propagation of context to resolve ambiguities.

In a later work [10], they focus on the segmentation of handwritten formuae, based on how human visually perceive mathematical expression. This segmentation uses horizontal and vertical projections, and groups symbols together. They use knowledge about mathematical symbols and syntax, and about how mathematical expressions are usually written. The relational tree obtained is used to recognize the structure of the expressions. The structure is recognized without knowing the identity of the symbols involved.

Winkler et al. [26] have a four-step approach. After a pre-processing step, they perform a *Symbol grouping* which basically handles structures like fraction lines, which contain a group of symbol above and a group below. Then comes the *(E.L.I.)-Classification* (for exponent, line, index), using relative vertical locations of symbols. For these two steps, they use a soft-decision approach, similar to fuzzy logic. Finally they generate the output, after a syntactic verification.

Genoe et al. [12] designed an online system using a fuzzy approach. Every time a new character is written, they recognize it with fuzzy rules and add it to an expression tree. Then, they identify the relationship it may have with other symbols using fuzzy rules based on the relative positioning of bounding boxes. Finally, they update the expression tree, using tree transformation rules.

In [2], Aly et al. are interested in the correct recognition of subscripts and superscripts. They use normalized bounding boxes as the main feature of a symbol. According to the type of symbol (ascender, descender or small), they add a virtual ascender and/or descender before trying to recognize the relationship. They proved that with normalized bounding boxes, along with a special treatment of irregular characters, they can very effectively identify the relationships, using a Bayesian classifier. Their work uses the *InftyCDB* data set, which contains expressions extracted from more than 70 articles, with different type-settings.

This work is extended to other types of relationships, such as *'above'* and *'under'*, in [1]. In this paper, the normalization of bounding boxes is explained in more details. Furthermore, they define more symbol classes. Whereas there are usually three main classes (ascender, descender, centered), they define six classes to handle variations in the positioning of some symbols. They still use Bayesian classifiers and get outstanding results on a large database. However, they assume that the symbol is always correctly recognized, and we can expect a lower accuracy if ever a symbol of one class is recognized as being a symbol of another class.

Conclusion Although there have been some early attempt to tackle the problem of mathematical expression recognition back in the 70s and 80s [25, 6], this topic really began to be researched in the 90s. A lot of papers have been published in the last 10 years, and we count 5 papers in this review, that are less than two years old.

Two facts can be noted:

- Character recognition is known to be difficult in the mathematical field, due to the huge number of different, but yet similar, symbols. However, very few published works (e.g., [25])consider that structure analysis could be performed beforehand. As far as we know, no author considers that a prior inference on symbol identity (or symbol class) can be inferred from the layout of the formula, even if Miller and Viola [17], Rhee and Kim [18] and Suzuki et al. [9] maintain ambiguities on the character recognition.
- Although symbol recognition is almost always performed using classifiers, only some papers consider the use of classifiers to identify the structure [1, 2]. Several papers use fuzzy logic, but most of them use either a baseline analysis, or graph-rewriting methods, or grammar rules.

Chapter 3

Motivations and Method Overview

In this chapter, we will explain the motivations for tackling the problem of mathematical expression recognition in a new way, in section 3.1. Section 3.2 is to define the scope of the project, which cannot include all kinds of mathematical expressions. Section 3.3 presents an overview of the method used to implement the proposed system according to our motivations. This method has been designed to work well within the defined scope.

3.1 Motivations and Hypothesis for a New Approach

3.1.1 Motivations

We explained in the previous chapter the problem of mathematical expression recognition. We have two motivations for trying a new approach. ***The structure of mathematical formulae looks simple enough to perform its recognition without the symbols' identities. The huge number of different symbols makes it difficult to recognize them directly.***

3.1.1.1 Apparent simplicity of the structure recognition

There are several reasons for performing the symbol recognition prior to the structure recognition. First, the identity of the symbols is a huge constraint on the possible structure. For example, the recognition of $\sum$ allows to define spatial regions for where arguments are to be found [21, 28] (e.g., above, below). Then, it is useful when one wants to understand the meaning of an expression. For example, the recognition of the triplet of symbols s, i, n on the same line allows the straight recognition of the 'sinus' function in the structure recognition phase, while 4 and 2 can constitute a number, and x followed by y an implicit product.

However, the symbols do not completely specify the layout. The rules for their association are well-structured. For instance, a superscript will never be found under its parent symbol, but always on the top-right position. The main component in the structure recognition is not the symbols but rather their positions and sizes. Some papers [2, 1, 9, 25] have shown that the use of the bounding boxes are often sufficient for the layout analysis.

3.1.1.2 Mathematical symbols are sometimes complicated

As we pointed out earlier, the range of symbols and rules used to write mathematical expressions is not fixed. The common symbols and structural rules are only a subset of a possibly infinite set, for symbols and new meanings for their association happen to be

invented. Indeed, new science areas keep appearing, bringing with them a need for new mathematical notation.

Quantum mechanics, for example, defined a new use of $<$ and $|$ to denote quantum states. The capital Greek letter Σ does not always denote a summation. The symbol $<$ represents a comparison between numbers in $1 < 2$, can be used as a kind of bracket in $\langle x, y \rangle$, or in the definition of some mathematical objects in $(\mathbb{C}, <)$. These are examples where existing symbols are reused with a different function. It is also possible to find completely new symbols.

Therefore, we should take this fact into account to recognize mathematical expressions. A convenient way of dealing with that problem is to give less influence to the symbol's identity. In most systems, the symbol recognition determines how the structure is recognized. A new symbol, which will not be recognized can provoke dramatic consequences in the structure analysis.

Similarly, when rules, written by hand, drive the structure analysis, the appearance of new relationships between symbols may compromise the recognition. For example, $\leq$ often represent an order between numbers. It usually have an argument on its left, and one on its right. However, in mathematics, this symbol is sometimes subscripted, as in $\leq_s$.

The spatial relationships between symbols are well-defined, in a finite number (subscript, superscript, and so on), but relationships can appear in a context in which they usually do not, as in the previous example. As a result, the recognition of relationships should be as less influenced by the symbol identity or class as possible.

3.1.2 Hypothesis

We believe that ***the symbols' identity is not necessary for the structure recognition***. Moreover, we think that ***the symbols can be classified using only their bounding box and the context available*** (Figure 3.1).

When we look at different symbols individually, we can see different spatial behaviours. For example, the relative position of a superscript, as we can see on Figure 3.2, will not be locally the same, when the symbol is a 'b' or a 'q'. At a global level however, all superscripts are supposed to be written on the same line (Figure 3.3).

If different classes of symbol produce different layouts, our hypothesis is the following. *If some layout is observed, it should be possible to tell which classes of symbol are likely to have produced it.* For example, we can estimate that the layout presented on Figure 3.4 corresponds to a descending symbol, subscripted, and followed by a small symbol.

To recognize it, we intuitively look at how symbols are positioned with respect to each other. A human would consider the whole layout to make a decision, but we think that the decision could be taken given a smaller amount of context. We call **context of a symbol** the information about the symbol itself (e.g. bounding box, class of symbol), but also about its parent and children. The important features are their relative size and position, and the kind of relation they have with each other. For example, in ab^c_d, the context of the symbol 'b' would contain:

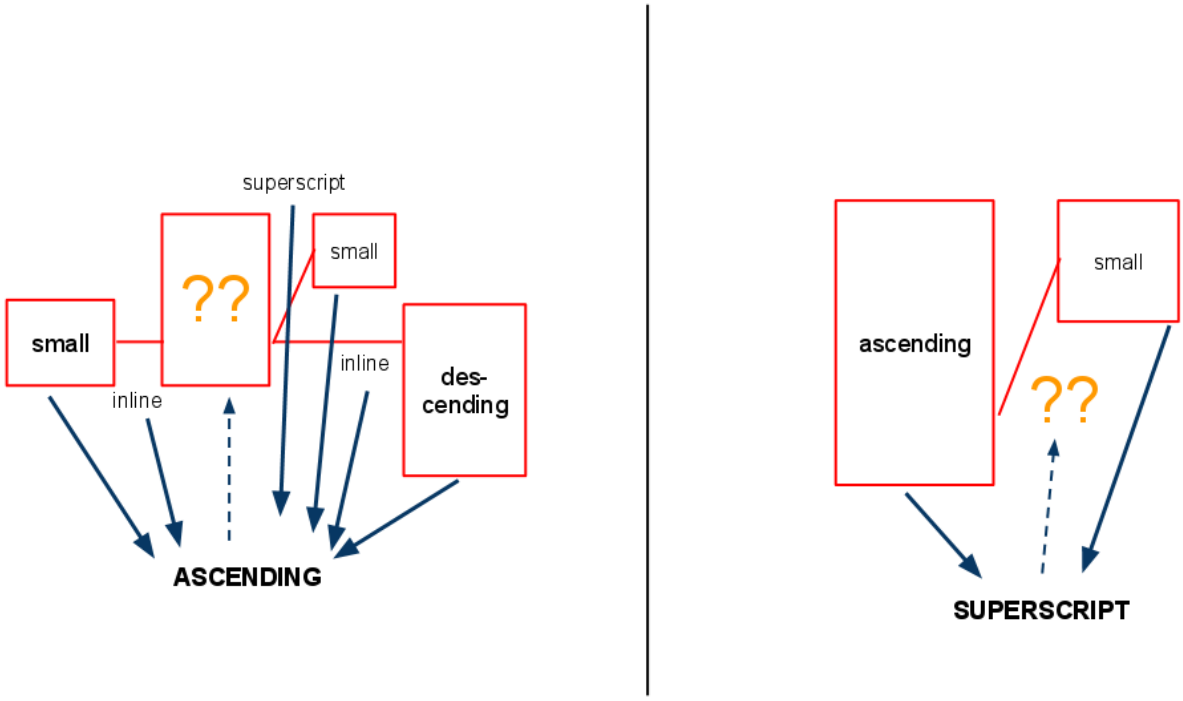

Figure 3.1: Hypothesis. Left: the context help classify the symbols. Right: the symbols' identity is not necessary for the structure recognition

Figure 3.2: Different relative position of a superscript

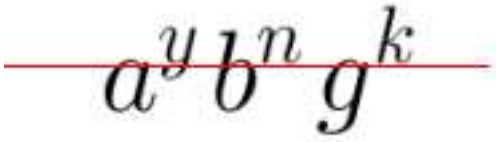

Figure 3.3: Arguments are on the same line

- Parent: 'a', inline
- Child: 'c', superscript
- Child:'d', subscript among other things.

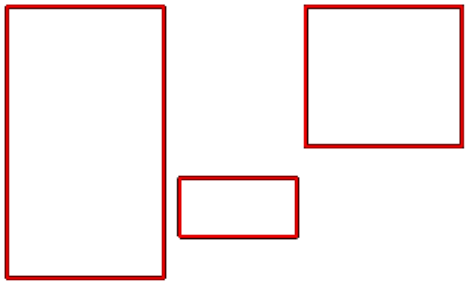

Figure 3.4: A Layout

We want to recognize the expression from the layout. For our analysis, we reduce the symbols to their bounding box. Recognizing the structure without any clue on the identity of the symbols is not an easy task. Indeed, different expressions, carrying different meanings can have the same layout. An example, taken from the survey by Chan et al. [5] is presented on Figure 3.5.

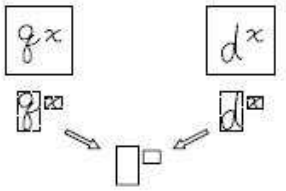

Figure 3.5: Similar layout (example from [5])

However, if there is a similar layout but we know the relationships between the symbols, it may help to disambiguate the expression, that is, to find the symbols classes. The example on Figure 3.5 shows two formulae with a similar layout: d^x and gx. If we know that x is a superscript in the first case whereas it is inline in the second case, we can infer that the first symbol is an ascending symbol in one example and a descending symbol in the other one.

3.2 Scope of the project

3.2.1 Complexity of Mathematical Expressions

Trying to recognize all kinds of formula is a huge task, so we will focus on mathematical expressions with limited complexity. We define the complexity of a formula in terms of order:

- **Order 0**: a formula of order 0 is only a one-dimensional sequence of symbols. It contains no subscript, superscript, etc. Some examples are given below.

$$a + b - N$$

$$\sum a \times \phi$$

- **Order 1**: a formula which contains only one level of nested structures. That is, for example, when subscripts and superscripts are expressions of order 0. Here are some examples:

$$a_p + b^{i+1}$$

$$\sum_{i=0}^{N} a_{i+1} + N$$

- **Order** n: formulae where nested structures are of order $n-1$.

Formulae of order 0 may look easier because one-dimensional, hence close the typical OCR. However, we use only spatial features in this project, so the amount of information in zero-order expressions is poor. Since we have more spatial complexity in first-order expressions, this might help to constrain the symbol class recognition, although introducing the issue of relationship recognition (e.g, we have to identify subscripts).

First-order formulae can turn out to be quite complex, as we need to identify a whole expression as a subscript for example. We define an intermediate order. A formula of **order 0.5** is a formula where nested expressions are individual symbols. For example, $a_b + c_d^e$ is a 0.5-order formula, whereas a_{b+c} is not.

3.2.2 Extent of the Project

The aim of the project is to create a method for the recognition, general enough to be adapted later to expressions that are outside the scope of this project. However, in a four-month project, it is crucial to define what will be done and what is left for a further work. This can be divided in two main points: the form of the input and its complexity.

The input is a binary image of a *printed mathematical expression*. We consider printed expressions rather than handwritten ones. First, they are easier to generate automatically. Moreover, there are less variations in the writing style. However, several techniques are used to ensure that the system can be later adapted for different inputs.

We aim to have good results for expressions of order 0.5. We call good results an almost perfect parenting and identification of the relationship between a symbol and its parent. We will also consider simple expressions of order 1 and 1.5, to test the robustness of our method and the ability of our system to adapt to more complex situations.

Not only want we to recognize the structure, but we also try to *find the symbol class using this structure*. A high order means a lot of context, which should make the classification easier. However, when expressions become more complex, the structure recognition becomes harder. A compromise must be done between the difficulties arising from the complexity and the need for context. Recognizing all kinds of symbols and all types of relationships remains a huge task. *We will focus on some classes of symbols and of relationships*. They are presented in the Tables 3.1 and 3.2, along with some examples.

3.2.3 Limitations

An MSc project cannot cover the whole problem. Consequently, there are also some limitations in the proposed system. We have to start from easy inputs, and make the system

Table 3.1: Symbol classes used

Symbol Class	Examples
1 - Small symbols	a, e, r, u, o, s, m, x, c, n, ...
2 - Descending symbols	y, p, q, g, ...
3 - Ascending symbols	A-Z, 0-9, t, d, h, k, l, ...
4 - Variable range symbols	$\sum$, $\prod$, $\bigcup$, $\bigcap$, ...

Table 3.2: Relationships used

Relationship Class	Examples
0 - Inline	xy, tan, 42, $10x$, $\sum n$, ...
1 - Superscript	p^n, b^a, x^y, ...
2 - Subscript	p_a, b_n, x_3, ...
3 - Upper	$\sum^{N}, \dots$
4 - Under	$\prod_{x}, \dots$

more complex step-by-step. The implementation must however handle basic mathematical expressions and should represent a system flexible enough, such that the proposed system should easily be extended in a future work.

First of all, we will not consider handwritten expressions, but only printed expressions generated with Latex, or with the Java API JLatexMath, used to render Latex expressions. However, since we use machine learning techniques, it is also possible to train the developed system with different inputs. We also do not consider non-connected symbols, such as 'i', 'j', '=', because we do not focus on the segmentation, and we impement a simple algorithm for this task.

We do not focus on the whole recognition. ***Our approach concerns the recognition of the structure, and how it can constrain the symbol recognition***. As a consequence, we are little interested in the identity of the symbols. For example, it is important that a 'p' is well identified as a descending letter, i.e. the confidence value for this symbol being in this class should be high enough. However, identifying the symbol p as the small letter 'p' is not in the scope of this dissertation.

As a result, understanding the meaning of an expression is not the goal of the project. For instance, we do not aim to recognize the tangent function in the formula $tan(\pi)$. Similarly, 42 will not be recognized as a number, neither will xy as an implicit product. Finally, the training sets will not contain all symbols for each class. We consider:

- in the class 'small', only small roman letters (e.g. 'a'),
- in the class 'descending', only descending roman letters (e.g. 'p'),
- in the class 'ascending', only ascending roman letters (e.g. 'b'), capital letters and digits,

- in the class 'variable range', only $\sum \prod, \bigcup, \bigcap$.

3.3 Method Overview

In this section, we will give a general presentation of the method used, and justify the algorithms implemented, and the design of the system. A thorough description of the design and the implementation can be found in the next chapters. First, we will show the different steps of the recognition in our approach. Then we explain each phase.

3.3.1 Presentation of the Proposed System

The input of the system is an expression in the form of a digital binary image. Since we focus on the structure recognition from the layout, too much information is contained in this set of pixels. We assume that it is sufficient to consider the bounding boxes of the symbols in order to represent the layout. Indeed, it gives us most of the useful information, namely the size and the position of each symbol. To yield this form, we need first to perform a segmentation of the image.

The output of the system is the recognition of the structure, and the prior recognition of the symbols. Three tasks are necessary to obtain this output.

Parenting symbols. Given a symbol, we have to know with which symbol it is in a relationship. For example, in a_b, a is the parent of b. Due to the nested nature of mathematical expressions, it seems more intuitive to talk about the parent of an expression. For instance, in a_{b^c}, b^c should be the child of a. We can avoid this problem, in way inspired by techniques which use minimum spanning tree to represent the relations between symbols (for example [21]). In an expression, we define a global parent. In b^c the global parent is b, and it is then the symbol used in the parenting with a in a_{b^c}. In the example a_{b^c}, a is the parent of b, and b is the parent of c. It can then be inferred that c is also in the subscript of a.

Identifying relationships. For each pair 'parent/child' of symbols, we must identify the relationship between the parent and the child. For example, in a_b, the relationship is 'subscript'.

Identifying symbol classes. Given the position and size of the bounding box, and the information obtained about relationships, this task must return confidence values for the considered symbol to be in each possible classes (small, descending, ascending, variable range). We can then deduce the symbol class. It is the class which have the highest confidence. For example, in a_b the first symbol (a) should be identified as a small symbol.

To parent a symbol with another one, we must see that there must be a relationship between them. Therefore, parenting symbols and finding relationships is one task. It is also very important to note that, in our approach, the recognition of a symbol class utilizes its context. We use the information about its parent and children. At the same time, as pointed out earlier, the nature of a relationship might not be clear if we do not know the class of the symbols. It leads to a chicken-egg problem, where each task need the results of the other one to be performed. To cope with that issue, we adopt an iterative method,

which consists in repeating each task, one after the other. However, this iterative process needs to be initialized. Thus, we perform first a rough symbol classification, using the only information available for each symbol before the structure recognition - the bounding box.

This process is summarized on Figure 3.6. The system is fed with a binary image, which is segmented. The bounding boxes help us perform a rough symbol classification. At this point, each symbol is associated with a class and some confidence values. It provides the necessary contextual information to try to identify the relationships. The context obtained is then used to refine the symbol classification. Since the symbol classes might have changed, we perform the relationship identification once again, and so on.

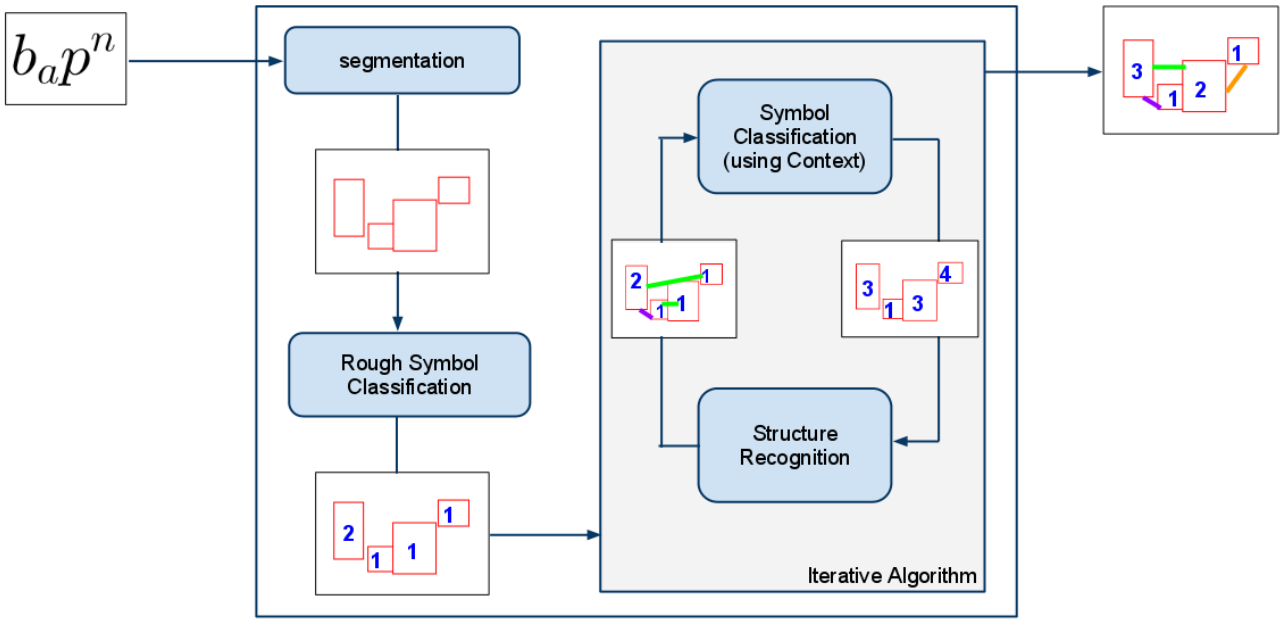

Figure 3.6: Overview of the Method

The result that should be yielded for the input $b_a p^n$ is presented on Figure 3.7. The parenting is represented by links between symbols. Each link is tagged with the relationship between a child and its parent: this is the relationship identification. Each symbol is tagged with its most likely class: this is the symbol class identification. Moreover, confidence values on the symbol class and on the relationships reflect the uncertainty of the system and the ambiguity of the layout.

3.3.2 Pre-processing

The goal of the pre-processing step is to build usable data. We want to keep it as simple as possible. There are two problems, namely how we represent the data and how we achieve such a representation.

The data will be composed of symbol representations. In an image, the symbol representation is a set of black pixels. Unlike humans, computers do not see the symbols

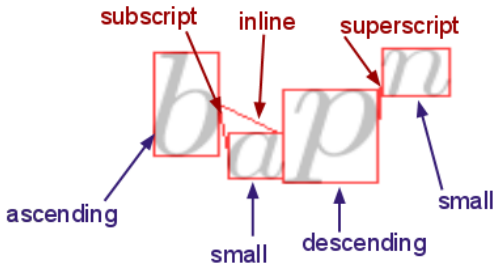

Figure 3.7: Example of Result

composed by these pixels automatically as a whole. We need a global information about symbols: a list of bounding boxes, associated with a list of spatial features. Therefore, we have to segment the image and return the list of bounding boxes.

Segmentation The requirements are the rapidity: the segmentation is not the core of our problem, and the robustness: most input images should be well segmented, for the data to be usable. We detect connected components [1] with a simple two-pass algorithm. Some symbols such as i, j or $=$ are not connected components. For the sake of simplicity, we ignore those symbols in our project.

List of symbols In our system, symbols are represented by their bounding box. Thus, the result of the segmentation is a set of symbols. To build the list of symbols, we use the fact that mathematical expressions are generally written left to right. Hence, we order the symbols according to the left bound of their bounding box. After this step, the expression is no longer an image, but a list of symbols (bounding boxes).

3.3.3 Symbol Classification

The symbol classification mainly considers two kinds of features. First is the information about the symbol alone, regardless of its context. This is the properties of the bounding box. Second is the context of the symbol. Since different classes of symbols produce different pattern for the position of the children, we try to find the most likely symbol class for the observed pattern. To achieve this task, we use several machine learning artifacts. Bayesian inference performs the rough classification. The classification using context is handled by artificial neural networks.

3.3.3.1 Rough Classification

The rough classification is the classification using no context whatsoever. It is based on the bounding box only. We call it rough classification, because the bounding box only gives us little information. Moreover, symbols that are in different classes can have the same box (see example on Figure 3.8).

Since the size, height, width and position of a symbol can vary from one input to another, we perform this classification using relative numbers. The parameter for the decision is the ratio width/height of the bounding box. The decision is based on the fact that symbols in

[1] A connected component is a set of black pixels in which any two pixels can be linked by a path containing only black pixels.

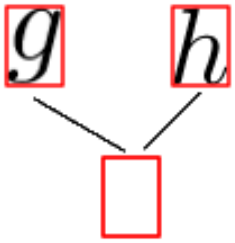

Figure 3.8: Example of Symbols with the same Bounding Boxes

the same class have similar bounding boxes. We can see this problem as a simple Bayesian network, presented on Figure 3.9. If the bounding box depends on the class, we can infer the class by observing the shape of the bounding box.

Figure 3.9: The Bayesian Network for the Ratio-Based Classifier

3.3.3.2 Classification using Context

Some available context, for example a subscript, can be used to make the classification easier. Indeed, the relative position and size of a child gives us extra information about the symbol itself. The idea is to have confidence values for a symbol being in each class, given the parameters associated with the context.

We can consider two main categories of context, namely the parent, and the children. The parameters associated with the context are:

- regarding the parent: the relative position and size of the symbol with respect to its parent, the relationship it has with the parent, and the class of the parent,
- regarding the children: the relative position and size of the child, the relationship (e.g. subscript) and the class of the child (e.g. descending).

It is important to note that the amount of context can vary. The leftmost symbol in an expression is considered as the start symbol and does not have a parent. It is rare that a symbol, especially a digit or a letter, have a full context, that is a parent, and a child for every relationship. The worst case is when the expression is only one symbol. This symbol have no context. The best case would be if every symbol had a child for every relationship. That is obviously impossible, otherwise the expression would have an infinite number of symbols. We have to handle this lack of context.

The parameters described before are inputs of neural networks which classify the symbol, and return confidence values for each class. Implementing a single neural network including the whole context would imply lots of missing values among the inputs. Moreover, several inputs correspond to the same piece of context, so we distinguish clusters of features. The role of one cluster would not be clear with a single network. Therefore, we opted for a system composed by several classifiers, one for each piece of context.

It makes a difference in the way the symbols are classified. We can see the single network as the symbol looking at its context and saying '*I am in this class!*', whereas the multi-classifier system would be each contextual symbol saying about the considered symbol '*This symbol is in this class!*'.

This approach enables a convenient managing of the lack of context. The different classifiers, including the rough classifier, work separately, but are used together. They are associated in order to return a reasonable result, as presented in the next chapter.

3.3.4 Structure Recognition

The structure recognition consists in associating symbols which are in relation with each other (parenting), and identifying the relationship.

Since we do not know the identity of the symbols for sure, we want to implement a flexible, adaptive method which returns confidence values rather than a crisp answer. We use a combination of a neural network to classify the relationship between two symbols, and of a score based on fuzzy baselines and fuzzy regions around the symbol. Moreover, a classifier tells whether a symbol can be a non-inline of a possible parent.

Aly et al. [2] proved that some parameters such as relative size and position allows to differentiate between subscript, superscript and on the same line. It is efficient when we know that the pair of symbols are in a relationship, which is not always obvious. As pointed out by Faure and Wang in [25], some symbols can be far apart but still in a relationship, especially for the 'inline' relationship. For instance, in $x^{superscript}y$, x and y are in the relation 'inline'. So we also consider the baselines to make these links between symbols on the same line. For the remaining parenting, we use fuzzy regions, similarly to Zhang et al. [29].

In the following, we will present these three artifacts, and explain how we use them together to recognize the structure. For more details about how it is really designed and implemented, report to the next chapters.

3.3.4.1 A Neural-Network based Relationship Classifier

This first classifier allows to classify the relationship existing between two symbols. It takes into account features inferred from a pair of symbols. Even if these symbols are not really in a relationship, it still classify the relationship. Therefore, it cannot say if the symbols are not in a relationship.

A neural network is used to implement this part of the system. As suggested by several papers ([25, 2]), the relevant features are the relative vertical position (D) and size (H) of the child with respect to its parent. These were proved to be efficient to discriminate between subscript, inline and superscript [25]. Aly et al. [2] showed that it was even

more efficient (99.89 % accuracy in the discrimination) when a virtual ascender and/or descender are added to the symbols which do not have one. This however requires to know the identity or class of the symbol. Since we do not have this information for sure, we chose the intermediate solution of adding as features:

- the symbol's class, and
- the parent's class.

As a consequence, we do not have to calculate new bounding boxes. It also puts the information regarding the classes at the same level as the other features. In [2], it implies a deformation of the bounding box, hence a modification of the parameters D and H presented above. Put differently, we can consider that for Aly et al., all bounding boxes look the same, and D and H for one pair can be compared with the same parameters for another pair, whereas in our technique D and H are somehow dependent on the symbols classes, hence we put the symbol classes as inputs of the network. This does not give the symbols classes a prior importance. Moreover, we consider the relationship 'upper' and 'under'. Thus, we also need a parameter translating the relative horizontal position. We created it, in a similar manner as D was designed in [2].

3.3.4.2 Finding Children in Class-Dependant Fuzzy Regions

Some authors, like Zanibbi et al. [28] perform the symbol recognition, and according to the symbol, define crisp regions around it for where to find arguments (e.g. on Figure 3.10, extracted from [28]).

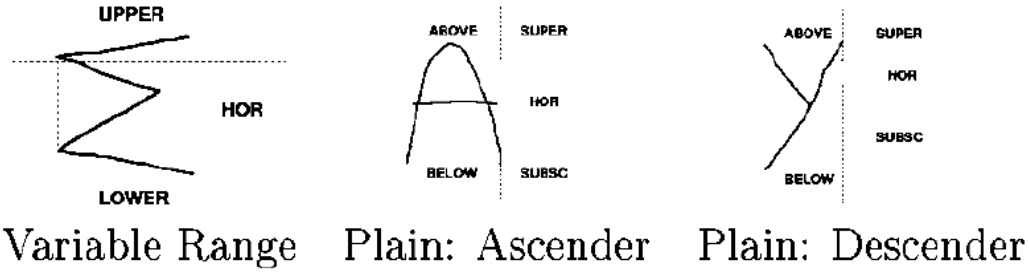

Figure 3.10: Examples of Symbol-Dependent Regions, from [28]

In a later work [29], they consider fuzzy regions for superscript, inline and subscript. We define as well class-dependent fuzzy regions. In this dissertation, the regions are built by hand, but it is possible to have the system learn them. To each class corresponds different regions. Since the class of the symbol is not assumed to be defined for sure, we mix the regions corresponding to each class, according to the confidence values for the symbol's class. We obtain regions which have two levels of fuzziness. One is due to the fact that for a given class, fuzzy regions are set. The other one is due to the uncertainty on the symbol class. We can see an example of how fuzzy regions are built (a) in [29] and (b) in our project, on Figure 3.11.

When we want to find a relationship between a symbol and a potential child, we compute the membership value for the child to be in each region. The scores obtained are combined with the classification of the neural network, to deduce the most likely relationship.

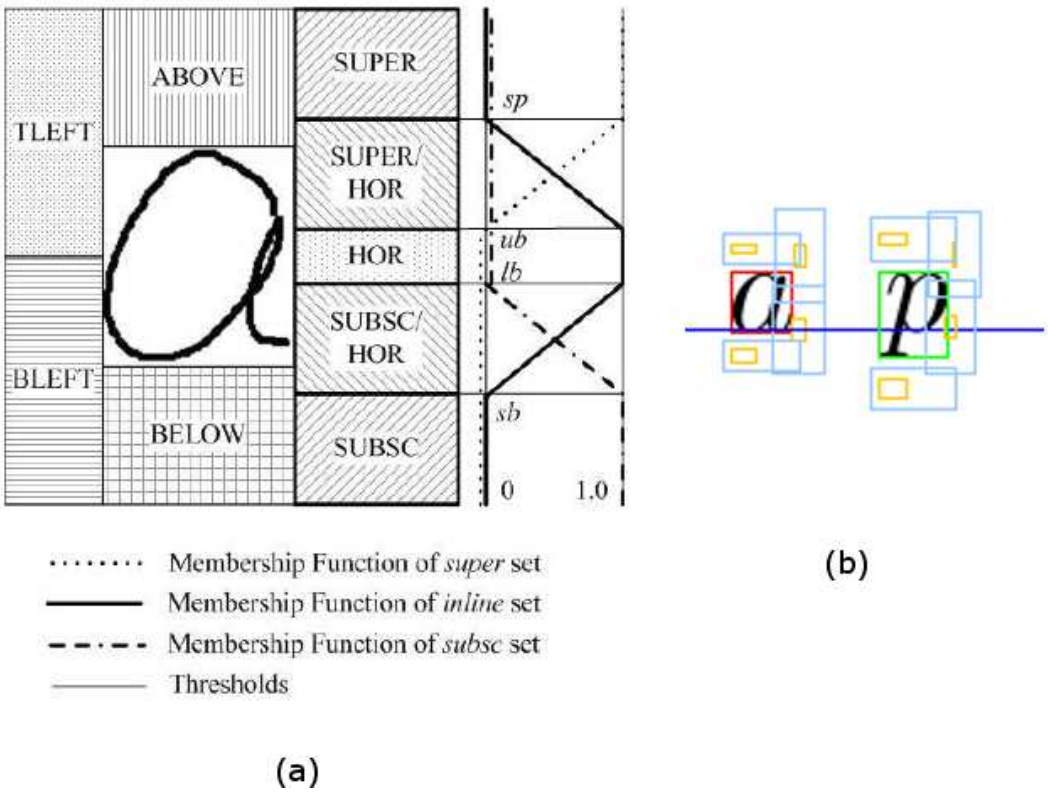

Figure 3.11: (a) Fuzzy Regions in [29], (b) Fuzzy Regions in our Project

3.3.4.3 Extracting Lines using Fuzzy Baselines

While 'superscript', 'subscript', 'upper' and 'under' often correspond to relationships between symbols close to each other, the relationship 'inline' is more complicated. Indeed, an inline child is seldom an argument of its parent, whereas it is often the case for the other relationships. It is very difficult to define a region in which an inline child should be found. We use a baseline analysis.

It looks like considering crisp baselines is sufficient for the project, because we use Latex printed expressions. However, we aim to build a flexible system. This project should implement a framework for a system able to recognize any kind of mathematical formulae, including handwritten ones. We use fuzzy baselines to handle small variations in the writing line. This has to be selective, because some nested baseline can be close to the dominant baseline, and we do not want the fuzziness to be a source of mistake. The fuzziness concerns the distance of the considered symbol's baseline to the baseline of its possible parent, but also includes the uncertainty about the symbol's class.

3.3.4.4 Recognizing the Structure

The recognition of the structure involves the parenting and the identification of the relationships. It relies on two observations. First, children that are not on the same line are close to their parent. Second, inline children can be far from their parent.

Two different treatments are applied for the parenting. Inline relationships are built based on the relationship classifier and on fuzzy baselines. Other relationships are based

on fuzzy regions and on the relationship classifier. We use an algorithm which is not recursive. One passage is sufficient for the parenting. We browse the expression left to right, assuming that children are never to be found on the left of their parent. Then, we implemented the parenting algorithm such that a limited backtracking is required to find the parent (see chapter 'Design').

The inherent difference between inline relationship and other relationships led to trying to find an inline parent before any other kind of relationship. The algorithm first checks if the considered symbol is aligned on an existing baseline. If not, it tries to find a parent, among the last symbols on each baseline. We also trained a classifier to determine whether two symbols can be in a non-inline relationship. We use the confidence for the class 'NO' in the score for 'inline', and for the class 'YES' in the other relationships' scores.

If N is the number of symbols in the expression, C its complexity, and R the number of possible relationships (here, $R = 4$), the worst case complexity of our algorithm is $2 \times R \times C \times N$. However, the algorithm is designed such that the complexity is usually lower. For more details about the algorithm, report to the next chapter.

3.3.5 An Iterative Process for the Whole Recognition

The symbol classification influences the relationship identification. But if the relationships change, it gives a new context for the symbols, and their classification might be different. In this case, the relationships may differ again. To translate these mutual constraints, we use an iterative algorithm, going back and forth from the symbol classification to the relationship identification.

Nothing ensures that a fixed point will be reached in the classification, because of ambiguities that might exist. Moreover, if a fixed point is reached, we cannot prove that the interpretation of the expression will be the right one. From these facts, two conclusions must be drawn. First, we should limit the number of iterations to avoid the problem evoked before. We cannot limit the number of iterations with a threshold on the confidence, because a high confidence does not mean a high correctness.

Most of the systems used in this project are based on machine learning techniques, i.e. neural networks or fuzzy techniques. We aim at a flexible system, which can adapt itself to different contexts. Besides, it should return fuzzy answers rather than crisp ones, because it is only a first step towards the whole recognition of the mathematical expression. Confidence values leave a choice for the final interpretation.

We create a data set which suits our problem. It contains expressions generated with Latex. To make the system more flexible, we add a Gaussian variation of the size and position of each symbol. We assume that it will reflect the variations existing in the different typesets, and in the handwriting, to some extent. When we train the classifiers, we prefer uncertain classifications rather than high accuracy. Indeed, the outputs will be considered as confidence values. An error is not a problem as long as the actual class has a high confidence too. Moreover, the classifiers are associated, and may correct each other's mistakes.

Chapter 4

Design

In the previous chapter, we explained the method we chose for building a system able to recognize the structure of simple mathematical expressions, and to classify the symbols using the layout. In this chapter, we focus on the practical design of our system. This includes the choice of the tools and machine learning artefacts, the setting of the parameters, and the algorithms used.

This chapter is made of three sections, organised as follows.

- Section 4.1 presents the preliminary work, needed to ensure a good design and an efficient implementation of the system
- Section 4.2 explains the design of the system, in terms of data representation, functionalities, design of the classifiers.
- The choice and design of the main algorithms will be shown in section 4.3. These algorithms are the segmentation of the input image, the recognition algorithm, and the exportation and processing of the results.

4.1 Preliminary Work

To design and build a system using machine learning techniques, some preliminary work is necessary. We have to have one or several data sets. Amongst other things, it allows us to analyse the data, which is another requirement to build a good system. We also have to choose the right tools to design and implement the system.

In this section, we will present these three tasks. First, we will discuss the tools we used, and justify their choice. Then, we will present the data-sets, how they have been constructed, and how we plan to use them. Finally, we will show the data analysis, the major prerequisite for the choice of the parameters.

4.1.1 Tools Used

In this part, we present the tools - software and frameworks - we have been using for the design and the implementation of the project. We will explain why they were useful, and how we used them. We classify them into three categories, corresponding to different steps of the project, namely, the analysis, the construction, and the implementation.

4.1.1.1 Tools for Analysis

For the analysis part of the project, we used mainly two tools: Matlab and Weka.

Matlab (Worcester College Licence) Matlab is a software for numerical computing and scientific simulation. It has its own language, based on matrices. This particularity allows to analyze the data easily, by plotting it for example. It can also handle images in a straightforward manner, and was used to test the segmentation algorithm.

Weka (GNU General Public Licence) Weka stands for Waikato Environment for Knowledge Analysis. It is a free software developed by the University of Waikato. It implements most machine learning techniques, and is an excellent framework to train and test intelligent systems, adjust their parameters, and find the best systems. It is composed of an API (Application Programming Interface) and a GUI (Graphical User Interface). Amongst other functionalities, Weka allows one to import data from a database, visualize it in function of the different data features, or even visualize classifier errors. This makes it an excellent tools for the analysis of data.

4.1.1.2 Tools for Construction

We call 'construction' the storage of the data-set, and the choice of the classifiers. To choose a good classifier, we have to test the parameters and analyze the results.

Microsoft Access We used Microsoft Access because it allows a straightforward creation and manipulation of data in tables using SQL (Structured Query Language). Moreover, Microsoft Windows provides an easy way to create an Open Database Connectivity (ODBC) with it. That made easy the interaction with Weka, which uses the JDBC (Java Database Connectivity) - ODBC bridge to access data.

Weka This time, the GUI of Weka was used to choose the classifiers, set their parameters, train and test them, and have a comprehensive description of their performance. This tool is fast and easy to test different classifiers, with different parameters, and be able to keep the best one.

4.1.1.3 Tools for Implementation

The implementation is the actual coding of the system. Due to the complexity of the program, we used several frameworks and APIs, such as JLatexMath, JDOM and Weka. The implementation was made in object-oriented Java, and we used Eclipse to program it.

Eclipse Eclipse is an Integrated Development Environment (IDE) which allows an easy and efficient programing and compiling in Java. It is a free, open-source software, released under the Eclipse Public Licence.

Weka In the implementation part, the API of Weka was used. All the functionalities of the GUI can be accessed through the API. It allows more flexibility. Since we want a system made of several classifiers, we can create, train, test, save and load them with the API, and implement the code for their association, in order to make them work together.

JLatexMath JLatexMath is a Java API, under a GNU Free Documentation License. It allows to render Latex expressions in Java. We use it to simplify the generation of inputs for the system.

JDOM (Java-based Document Object Model) According to the project's website, "JDOM is available under an Apache-style open source license, with the acknowledgment clause removed" [1]. It is a very simple API to read and write XML (eXtensible Markup Language) documents. It is used in our system to save the interpretation of an expression, in order to build a test set. The possibility of reading XML document is used to load the test set and compare the expected results to the actual recognition.

4.1.2 The Data Sets

As for most machine learning based systems, we need a data-set. For it to be used in our project, it must contain the following information:

- size of the formulae and of individual characters,
- position of the characters in each formula,
- identity (or at least class) of each symbol, and
- relationships between symbols.

Suzuki et al. [19] released such a data-set, very comprehensive. It contains 20.767 mathematical formulae, and more than 157.000 symbols, taken from 30 scanned scientific articles. We will present it in a first part.

However, the comprehensiveness can be a problem. In this project, we want to begin with very simple formulae to first test the system on simple cases. Introducing more complex structures should come afterwards, and very complex expressions are not in the scope of this dissertation. In the second part, we present a way to design a simpler data-set, which should satisfy our initial needs.

4.1.2.1 Overview of an Existing Data Set: $InftyCDB-1$

This data-set has been constructed by Suzuki et al. for the Infty project [19]. This project is devoted to the recognition of mathematical formulae. The data set consists of characters information along with some features such as their bounding box, their size, the relationship with their father symbol, and so on. It has been built from 30 scientific papers published in English. There are characters extracted from both plain texts and mathematical formulae.

There are 175 different symbols, and 8 kind of relationships, namely, *horizontal*, *left subscript*, *right subscript*, *left superscript*, *right superscript*, *top*, *under* and *upper*. A Latex, MathML and IML equivalent is given for each formula. The data-set can be freely downloaded on the project website (`http://inftyproject.org/`). The data is provided in CSV and MS Access format, and scanned images are provided too.

[1] from www.jdom.org

As explained in the previous chapter, our goal is not to recognize all kinds of expressions. We set a limit on the complexity of the formulae we aim to recognize. Moreover, we chose to focus mainly on Latex formulae. It is difficult to extract efficiently from this data-set the relevant parts to our project.

4.1.2.2 Towards a New Data Set

The aim of building a new data-set is to avoid being overwhelmed by the complexity of the existing ones. By doing so, we can consider only the symbol classes and relationships in which we are interested for this dissertation. The advantage is that we have a total control on the content of the data set, the features it contains, and the complexity of the expressions it represents.

The drawback is that building a data set often involves the manual labelling of its content. A good data set should contain as much examples as possible. Manually label thousands of expressions requires too much time for the length of this project.

Therefore, we had to find a way of building the data set automatically. This means that the examples must be simple enough to be labelled by a program. The simplicity might not represent well enough what mathematical expressions are like in the real world. However, we should remember that the data set will be used to train the individual components of the whole system. These components do not focus on the recognition of the whole expression, but of small parts of it. For example, the relationship classifier only consider two symbols, independently of the rest of the formula.

Thus, the simplicity of the examples should not be an issue, as long as it represents the context in which the symbols will be found. Since we control the way the expressions are produced, we can exploit their writing order. The pipeline for the data set creation is presented on Figure 4.1.

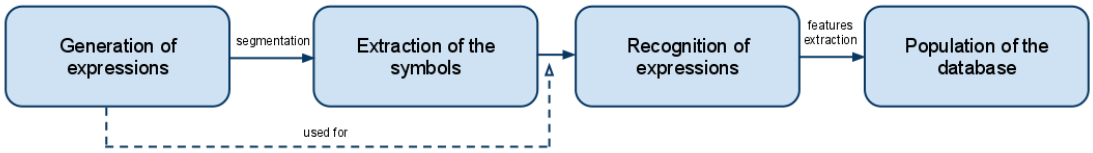

Figure 4.1: Creation of a New Data Set

a. Generation of Expressions

The process of expression generation will be used for the recognition of the expressions. Therefore, we have to generate non ambiguous formulae, which recognition can be straightforward without any classification. It means that after the segmentation of the image, we can be able to tell the class of symbols and their relationships, with the mere knowledge of how it was produced.

Thus, we use a systematic approach for the symbols classes and the relationships, with a simple reading order. We have four classes of symbols. For each class, we choose a certain number of symbols (see Table 4.1).

Table 4.1: Symbols used in the Data Set

Symbol Class	**Symbols used**
small	a, z, e, r, u, o, s, m, x, c, v, n
descending	y, p, q, g
ascending	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, Z, E, R, T, Y, U, I, O, P, Q, S, D, F, G, H, J, K, L, M, W, X, C, V, B, N, t, d, h, k, l, b
variable range	$\sum$, $\prod$, $\bigcap$, $\bigcup$

The reading order associated with the relationships can be of two kinds:

- mainly horizontal: inline, subscript, superscript
- mainly vertical: upper, under

They should be associated with two different generation processes.

For the generation of horizontal structures, we use the classes 'small', 'descending' and 'ascending'. We produce all possible triplet of symbols for these classes and the relationships in a systematic manner. Each time we choose randomly which symbol of the considered class will be represented. Each pattern 'class, relationship, class, relationship, class' is used twice to have more examples. An extract of the generated data set is shown on Figure 4.2.

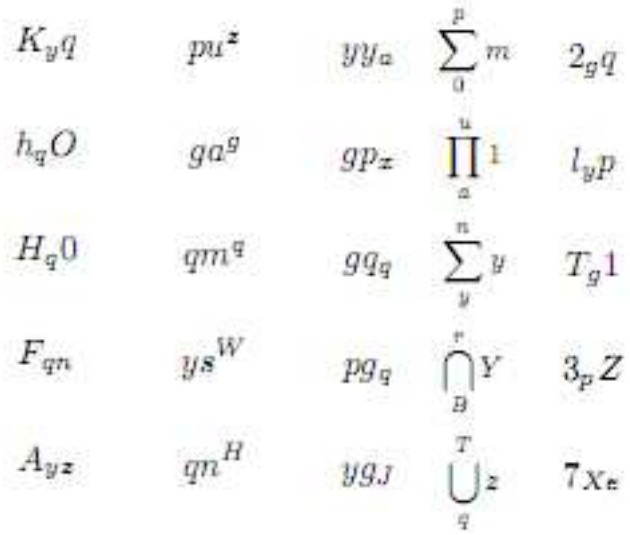

Figure 4.2: Examples in the Data Set

A similar method is used for vertical structures. We generate a variable range symbol, and a symbol above, one below and one inline. The last three are alternatively chosen in each class but 'variable range'. Finally, variable range symbols can be children of other symbols (for example, in $x^{\sum i}$). We also created this type of examples, in a similar manner. Images are generated using the jLatexMath framework, and are the inputs of the next step. The complete algorithms for the generation of examples are presented in the appendix 'Algorithms'.

b. Extraction of the Symbols

The symbols are extracted from the images using our segmentation algorithm. This step returns an ordered list of bounding boxes, representing the symbols of the expression in our project. This is the input of the next step.

c. Recognition of Expressions

In the first set of examples, we are supposed to retrieve three symbols, left, middle and right. According to the generation process, we can label them with their class. We also do the parenting automatically, and label the relationships. In the second set, the same logic applies. From the bounding boxes, we detect a leftmost symbol, which is the variable range symbol, a top and a bottom symbol, and a rightmost symbol. We can then automatically label the symbols and relationships.

The recognition is entirely driven by the method with which the examples are initially created. The recognition of the expressions is then completely deterministic, and no mistake is possible. The information about the symbol (position, size and class) and about its context (parent, relationship, children) is stored at the symbol level (see the 'Implementation Overview' chapter), and allows each symbol to create its instance in the database.

d. Population of the Database

In the database, each line corresponds to a symbol. In order to train the classifiers of the system, some features must be available, hence calculated when we create the database. The position and size are directly deduced from the coordinates of the bounding boxes. The class of the symbol, its parent and children are inferred from the automatic labeling. Some features have to be calculated, such as the ratio width/height of the symbol, its relative position and size to its parent, and the relative position and size of its children. The relevant features are listed in Table 4.2, along with their definition.

The way Latex generates equations obeys some structural rules regarding the baselines and the size of the symbols. Each instance in the data-set is different, due to the generation process. It uses different symbols randomly chosen in the classes, and different relationships and arrangements. But we can still consider it redundant, because we can find similar symbols with similar contexts. For example, $a^b c$ and n^{b_x} are different, but in the database, the line corresponding to a and the one corresponding to n are likely to be very similar.

Table 4.2: Features in the Data Set

Feature	**Description**
XMIN	Left bound of the bounding box
XMAX	Right bound of the bounding box
YMIN	Top bound of the bounding box
YMAX	Bottom bound of the bounding box
HEIGHT	Height of the symbol
WIDTH	Width of the symbol
RATIO	Shape of the bounding box
H	Relative height of the symbol to its parent
D	Relative vertical position to the parent
V	Relative horizontal position to the parent
RELID	Relationship with the parent
CLASS	Symbol class
PCLASS	Class of the parent symbol
LH	Relative height of the inline child (if any)
LD	Relative vertical of the inline child (if any)
LV	Relative horizontal position of the inline child (if any)
LCLASS	Class of the inline child (if any)
EH	Relative height of the superscript child (if any)
ED	Relative vertical of the superscript child (if any)
EV	Relative horizontal position of the superscript child (if any)
ECLASS	Class of the superscript child (if any)
SH	Relative height of the subscript child (if any)
SD	Relative vertical of the subscript child (if any)
SV	Relative horizontal position of the subscript child (if any)
SCLASS	Class of the subscript child (if any)
UppH	Relative height of the 'upper' child (if any)
UppD	Relative vertical of the 'upper' child (if any)
UppV	Relative horizontal position of the 'upper' child (if any)
UppCLASS	Class of the 'upper' child (if any)
UndH	Relative height of the 'under' child (if any)
UndD	Relative vertical of the 'under' child (if any)
UndV	Relative horizontal position of the 'under' child (if any)
UndCLASS	Class of the 'under' child (if any)

This fact, combined with our willingness to have a somewhat flexible system, motivated another choice in the data set conception. To simulate variations within expressions, we modify slightly the position and size of each symbol. This is done after the recognition of the expression and before the calculation of the features, to remain consistent. We recompute the coordinates, the height and the width of the symbol, according to a random variable corresponding to a draw from a Gaussian. The parameters are presented in Table 4.3.

The data-sets were generated by outputting a Comma-Separated Values (CSV) file, used to populate a database.

Table 4.3: Gaussian Variation

Parameter	Mean	Standard deviation
x-coordinate	original x-coordinate	5% of symbol width
y-coordinate	original y-coordinate	5% of symbol height
width	original width	5% of symbol width
height	original height	5% of symbol height

Conclusion We developed a method to create an automatically labelled data set, which was used to train the different classifiers of the system. It was useful for the context-driven classification of symbols, but also for the relationship classifier. The variation introduced in the size and position of symbols also allowed us to classify some symbols and relationships in an handwritten expression during the tests, even though it was not in the scope of the project. Overall, the data set contains 7316 instances, automatically generated. We also created a data set for training the rough symbol classifier. It was built by generating individually the Latex version of each symbol used. The features are only the ratio width/height of the symbols and their class. Each symbol appears six times in the data-set, with different Gaussian variations. This second data-set contains overall 372 instances. A third data set is created using the examples of the first data set, to train the classifier saying whether two symbols can be in a non-inline relationship.

4.1.3 Data analysis

Along with the data set, the analysis of the data is the crucial element of the construction of a system based on artificial intelligence artefacts, and machine learning techniques in particular. It consists in looking at the data and features from different point of view. For example, we can plot the expected classification of data instances, as a function of several parameters.

This analysis has several goals. Amongst them, we can cite:

- verifying assumptions on the data,
- making sure that our motivations and hypothesis are worth investigating,
- spotting unexpected behaviours of the data,
- choosing relevant features for the building of the classifier, or even
- understanding the possible bad performance of some classifiers.

Since the mathematical expression recognition is made of two tasks, we can split the analysis into two categories. The first one concerns the classification of relationships. The second one focuses on how symbols can be classified using their context.

4.1.3.1 Analysis of the Data on Relationships

The analysis of the data regarding relationships aims at proving that the features we chose allow (i) the identification of the relationships and (ii) the building of fuzzy regions.

The parameters we use for the classification of relationships (H and D) have already proved to be good by [2]. However, we replaced the knowledge about the parent symbol's identity present in [2] by the knowledge of the classes of the symbols involved.

Figure 4.3 shows that inline, superscript and subscript relationships can be discriminated using these extra features.

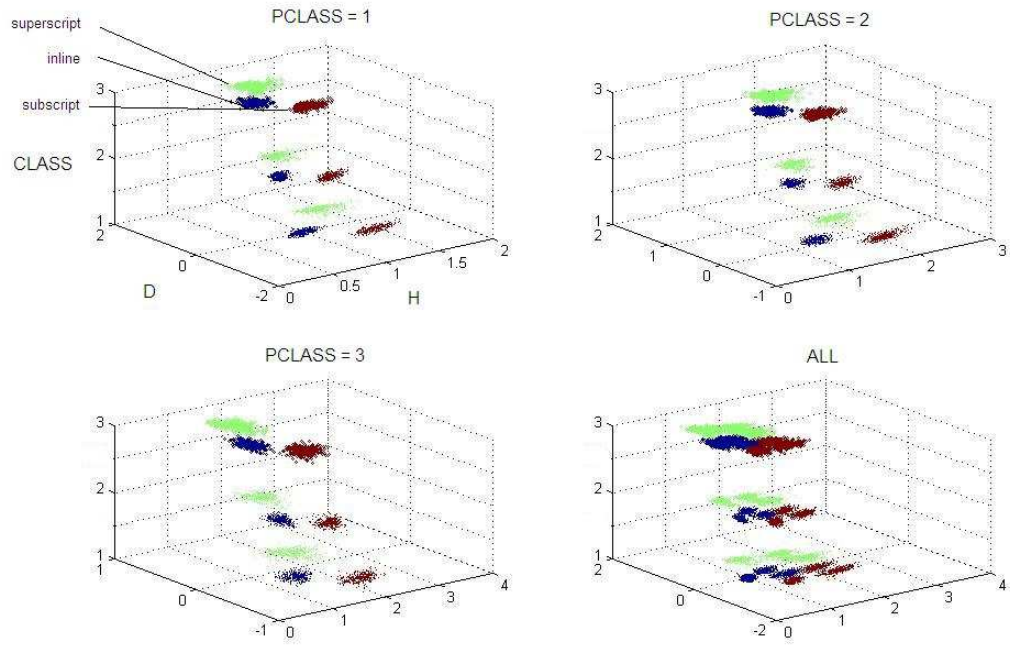

Figure 4.3: Relationship Data in the Space ($PCLASS$, $CLASS$, H, D)

Moreover, for each parent class, there appears to be an area where all possible subscripts (resp. superscripts) are positioned. This should make possible the construction of fuzzy regions based on the data set.

4.1.3.2 Analysis of the Data on Symbol Classes

The analysis of the data regarding the symbol classes aims at confirming some of our hypothesis. We use only bounding boxes. The first part shows that the shape of the bounding box, although giving a first idea, is not sufficient to classify symbols. The second part proves the importance of considering the context of the symbol.

The Ratio Width/Height

On Figure 4.4, we can see the values for the ratio width/height of the symbols in the data set. We can see some areas where the classification should not be a problem. A ratio over 1.5 corresponds to small symbols (class 1), and to ascending under 0.6. However, the ascending class seriously overlaps the other ones, due to the high variety of symbols in this class. But we can still infer some rules, presented in Table 4.4.

We conclude that a classification based on probability can give us a good but far from perfect idea of the possible symbol class, and cannot be used alone.

The Importance of Context

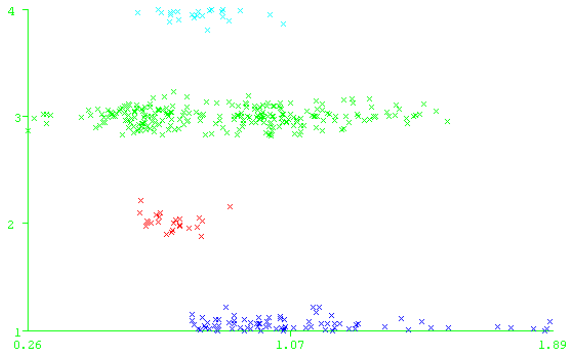

Figure 4.4: Ratio of Symbols

Table 4.4: Class Areas Based on Ratio

Ratio	**Possible Class**
< 0.6	ascending
$0.6 < .. < 0.75$	descending, ascending or variable range
$0.75 < .. < 1.5$	small or ascending
> 1.5	small

We tried to train a single classifier based on all features for the symbol classification. When we analysed the results, we noticed that most classification errors correspond to the symbols which have little, if any, context. This leads us to building one classifier corresponding to each piece of context. On Figure 4.5, we represent the data for superscripted symbols in the space (EH,ED) (see previous section for explanations).

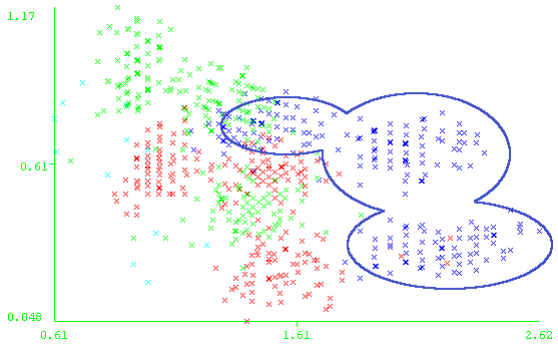

Figure 4.5: Data for superscripted symbols

The data points can already be clustered using these features. If we also consider the class of the superscript ($ECLASS$), there is almost no overlap. Two conclusions can be drawn:

- the context appears to help the symbol recognition, and
- a good classification should result from the separation of the classifiers according to which part of the context they represent.

We have to note that this analysis considers the parameters to be right, especially regarding symbols' classes (e.g. $ECLASS$, or $PCLASS$) and the relationships. In the

actual classification, some errors might result from mistakes in the symbol classification step, or in the structure recognition.

4.2 Design of the System

The design of the system includes different choices:

- the way the data (from input to output) is represented
- the way the components are associated to build the multi-classifiers
- the values of the parameters and thresholds
- the functionalities we want the system to have, other than the simple recognition of expressions
- the way we want to display the information to the user

In Section 4.2.1, we explain how we represent the data. Section 4.2.2 presents the design of the classifiers and their association. In Section 4.2.3, we present the design of some additional functionalities of the proposed system. Finally, in Section 4.2.4, we introduce the design of the user interface.

4.2.1 Representation of the Data

During the process, the form of the data changes. Indeed, the input is a binary image, and the output should be an interpretation of the expression. The different steps of the recognition have been identified. We must decide how to represent the data at each stage. In this part, we will present the data representation in a chronological order, starting with the form of the input, and finishing with the form of the output, as summed up on Figure 4.6.

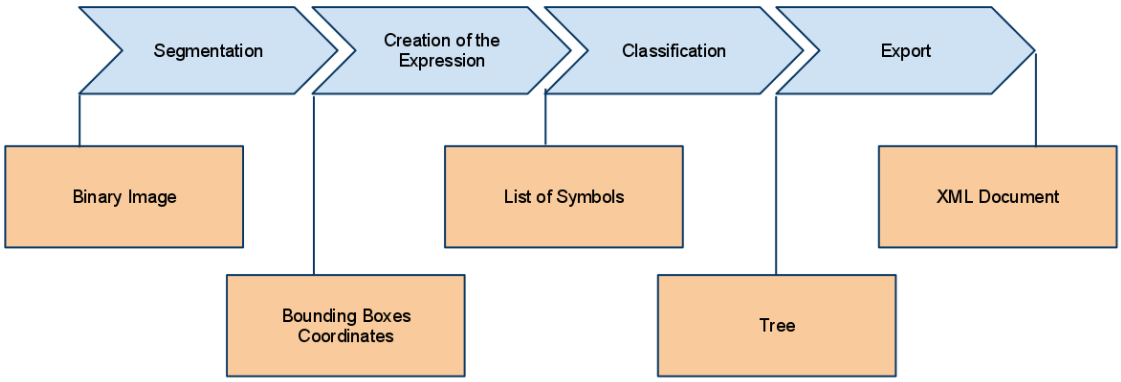

Figure 4.6: Data Representation along the Pipeline

The input: a binary image

We chose to take binary images (or bitmap) as inputs of the system. It represents the context in which mathematical formula recognition is needed. It corresponds to the case in which we have a printed (or handwritten) document which contains a formula that we want to extract to obtain an electronic form, which can later be reused.

A binary image is merely a sequence of bits, which convey an information to the human eye when they are displayed in a visual form (an image). The information a computer gets by simply looking at the sequence is however limited. This is why we transform this data, by segmenting the image.

Result of the segmentation: a list of bounding boxes

The segmentation algorithm reads the image and extract connected components from it (see the 'Presentation of the Algorithms' section). From the connected components found, we only keep the bounding boxes. This is indeed the simplest way to represent the layout of the expression. Each item in the list of bounding boxes is in the form $x_{min}, x_{max}, y_{min}, y_{max}$. In other words, it is the left, right, top and bottom bounds of each box.

A list of symbols when the expression is created

The list of bounding boxes is used to create the initial representation of the expression, which is merely a list of symbols. The structure of the expression is presented on Figure 4.7.

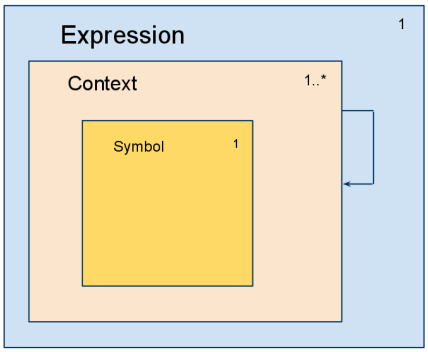

Figure 4.7: Representation of an Expression

The *'Symbol'* entity represents the symbol only, out of its context. It is created from the coordinates of the bounding box. It contains information about the size and position of the symbol.

The *'Context'* entity represents the symbol in its context. It can be linked to other symbols (Contexts) such as the parent of the symbol or its children. It also contains the relationship the symbol has with its parent, and the confidence values on the class of symbol and on the relationship.

The *'Expression'* entity is made of the list of symbols (Contexts).

The creation of the expression builds a 'Symbol' object for each bounding box in the list, and create a 'Context' for each symbol. All contexts are stored in a list, which is the 'Expression'.

The classification builds a tree

The aim of the classification and the recognition is to find the relationships between symbols, and to find the symbols classes. This can be seen as linking symbols together, and adding information to the existing structure. From the initial list, a tree is created. Similar tree representations can be found in the literature (such as [21, 9, 28]).

Each node corresponds to a context. As explained before, the information contained in a node is:

- the corresponding symbol
- the relationship with its parent
- the regions where children should be found
- the distributions of confidence values over the possible symbol classes, for each classifier (see next part)
- the distribution of confidence values over the possible relationships with the parent

Each child of a node corresponds to a child of the symbol represented by this node. There is also a backward linking to access the parent of the symbol. The tree representation is summed up on Figure 4.8.

A partial example of such a tree is showed on Figure 4.9, for the expression $b_a p^n$.

Exporting the results as an XML document

Once the classification is done, we want to be able to save the result. There are several goals for that. First, it could be used by another program to finish the recognition. Indeed, we classify symbols and recognize the structure, but some work must still be done. For example, the recognition of each symbol, or even the equivalent of the lexical pass described in [28]. The result saved can also be reused by the program, to load the interpretation of a formula for example.

We have chosen an XML form, because it translates well the nested structure of a mathematical expression. Moreover, an XML document is a tree structure, like our expression representation is. The document must contain the important information:

- The expression, its width and height
- The symbols, their class, relationship, bounds, and the confidence values for each class and relationship

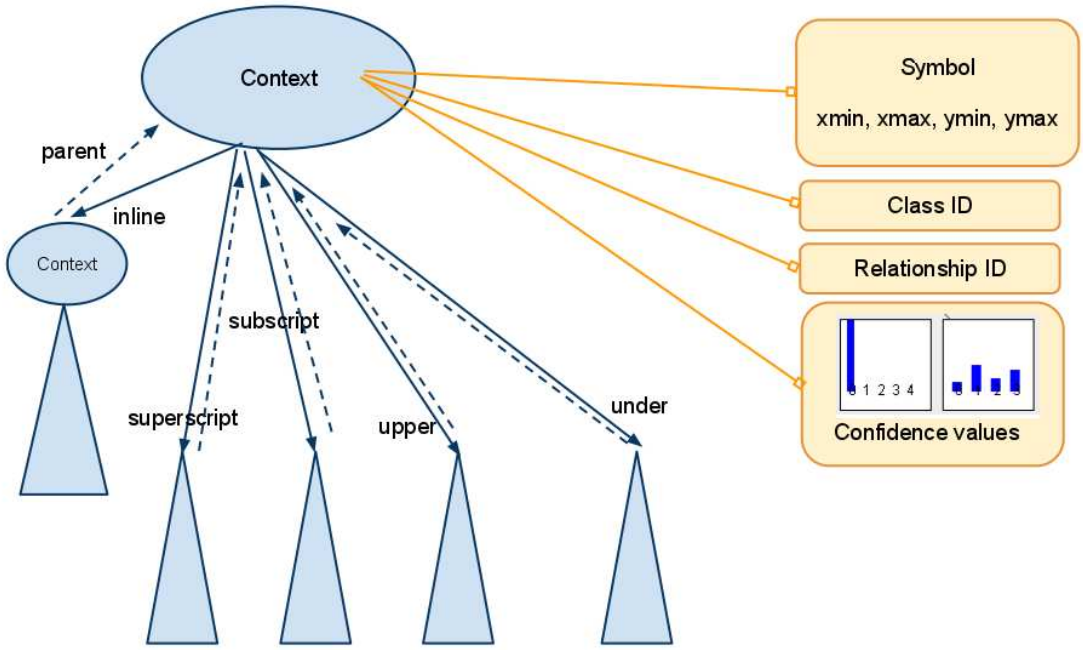

Figure 4.8: Expression Tree

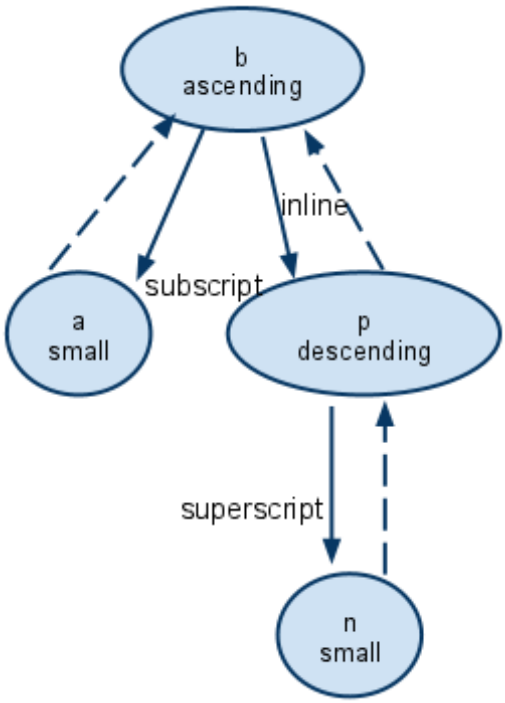

Figure 4.9: Example of an expression tree

The root of the document is the expression node. It has only one child which is the representation of the root of the expression tree. It is a symbol node. A symbol node has three attributes:

- The id attributed in the creation of the expression
- The symbol class id
- The relationship id

The symbol id is used when the XML interpretation is loaded by the program, in order to be able to identify the correspondence between a node in the expression and a node in the document. It has several children:

- A bounding box node, which contains the coordinates
- A symbol class node, which contains the confidence values for each class
- A relationship node, which contains the confidence values for each relationship
- A symbol node for each child

An example is shown on Figure 4.10.

```
<expression width="126" height="129">
      <symbol id="6" class="1" rel="-1">
            <boundingBox>
                  <xmin>43</xmin>
                  <xmax>65</xmax>
                  <ymin>63</ymin>
                  <ymax>85</ymax>
            </boundingBox>
            <symbolClass>
                  <class id="1">0.4303918844197614</class>
                  <class id="2">0.1428140109883948</class>
                  <class id="3">0.26488460479316966</class>
                  <class id="4">0.16190949979867414</class>
            </symbolClass>
            <relationshipClass>
                  <class id="0">0.0</class>
                  <class id="1">0.0</class>
                  <class id="2">0.0</class>
                  <class id="3">0.0</class>
                  <class id="4">0.0</class>
            </relationshipClass>

            <!--CHILDREN-->
            <symbol id="5" class="3" rel="1">
            ...
            </symbol>
            <symbol id="3" class="2" rel="0">
            ...
            </symbol>
            ...

      </symbol>
</expression>
```

Figure 4.10: Expression Interpretation in XML

4.2.2 Practical Design of the Classifiers

The part of the system that actually recognizes the expression is made of classifiers, such as neural networks, or fuzzy systems. The system is an association of these different entities. Each component is designed for a particular role. We have to choose the best suited system to represent it, and adjust the parameters, define how it will be trained for example, or also which kind of results is expected. We also have to choose how these

different components are associated together, how their outputs are mixed to obtain the final result.

The system is made of two parts, one for each task. We will present first the Symbol Classifier (SC), and then the Relationship Classifier (RC).

4.2.2.1 Symbol Classifier

Its aim is to classify the symbols into one of the four classes 'small', 'descending', 'ascending', 'variable range'. It would not be appropriate to give a straight answer because we consider only the bounding boxes. Rather, we should return the confidence values for the symbol to be in each class.

According to our plan, we want to consider the context of the symbols to classify them. We will use a system composed of several classifiers. Indeed, having only one classifier poses two problems:

- At the first step of our algorithm, we perform a first symbol classification when no context is available, so we have to have a classifier which does not take the context into account.
- All symbols have not the same amount of context. Some symbols might have very little context. For instance, in $b_a p^n$, n only have a parent, but no child. If we have a single classifier, the absence of context may be understood as an important information, when it is rather a lack of information.

We will first present and justify the choice of the multi-classifier. Then, we will explain how the parameters are chosen, and how the components are trained.

Form of the classifier

The system chosen is a multi-classifier which can be adapted to each symbol, given its context. The different classifiers each returns a set of confidence values which are then merged together. The system is summarized on Figure 4.11.

The output is a mix of the ratio based classifier, the parent based classifier, and the children based classifier. This last one is made of five classifiers, one per possible child. In the following, we will present each classifier briefly. Then we will give a justification of the general form.

Children based Classifier. The children based classifier is made of five artificial neural networks, one for each possible child. The 4 inputs of each classifier are

- the class of the child,
- its relative vertical position,
- its relative horizontal position, and
- relative size to the parent.

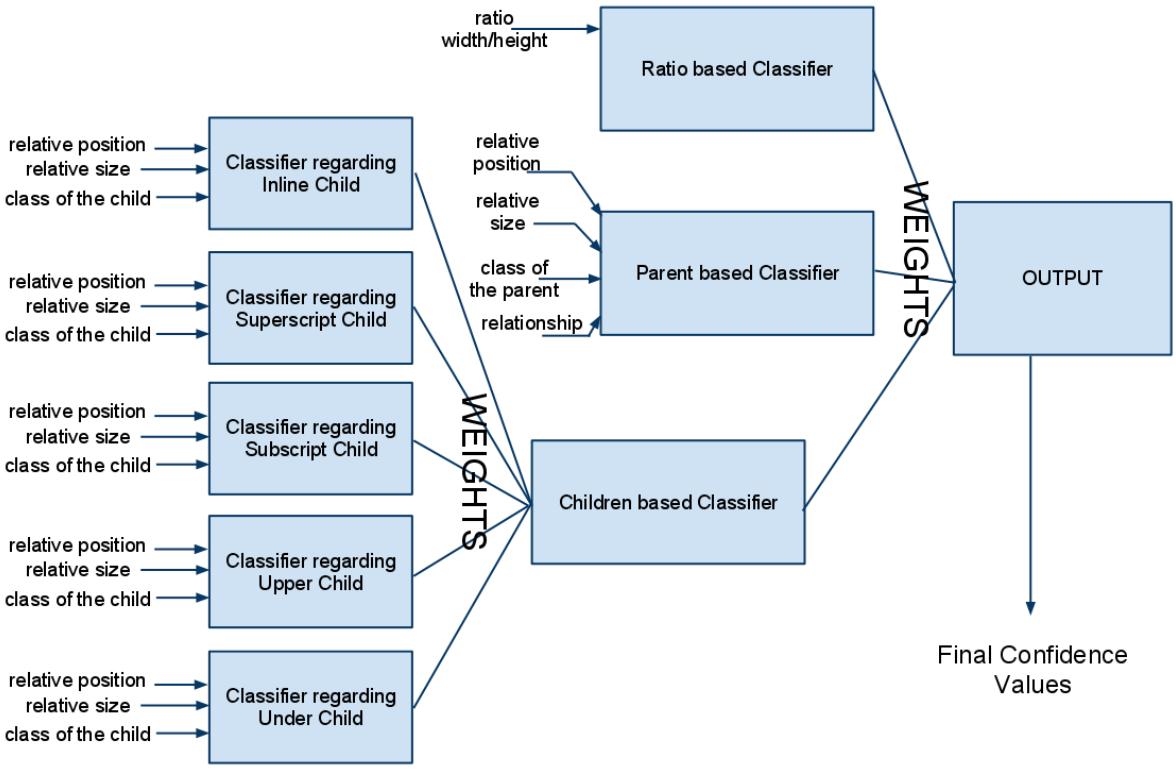

Figure 4.11: The Symbol Classifier

Parent based Classifier. The parent based classifier looks at the relative position and size of the symbol to its parent. It also takes into account the parent's class, and the kind of relationship (e.g. subscript) the symbol has with its parent. It is also an artificial neural network.

Ratio based Classifier. The ratio based classifier allows to classify the symbol regardless of its context, using only the bounding box information. It is particularly useful for the first step of the classification: the rough symbol classification. Indeed, at this stage, no parenting is done, and no relationship is identified. We use a Bayesian inference system. We assume that the ratio (or shape of the bounding box) is a consequence of the symbol class. Indeed, symbols of the same class are likely to have a similar shape. For example, 'small' symbols, such as 'a', 'n' or 'm' generally have a ratio width/height equal or higher than 1.

Justification of the form

The whole classifier is divided into several classifiers, associated as presented above, because:

- The context is a variable amount of information, which must be handled in a particular way (see next paragraph).
- It is crucial to give different importance to different pieces of information. For example, if we had a single classifier, we would not have very much control on the predominance of the ratio information over the class of a superscript for example.

The ratio has a strong correlation with the symbol class, whereas the child class is not directly dependant on the symbol class. We clustered together the features which make sense only when they are taken together, e.g the information about the subscript is different from the one about superscript, and the information about the parent, the children and the ratio are different kinds of information.

- The last reason was that we needed a component for the rough symbol classification, when no context is available

Handling the lack of context

The system is made for the classification of a symbol in a full context. However, in mathematical expressions, most symbols have an incomplete context. We thought of two ways of handling this lack of context. First, we can ignore the classifiers which correspond to a non-existing context. The system for a symbol which has no parent, and only a superscript, would then be as presented on Figure 4.12.

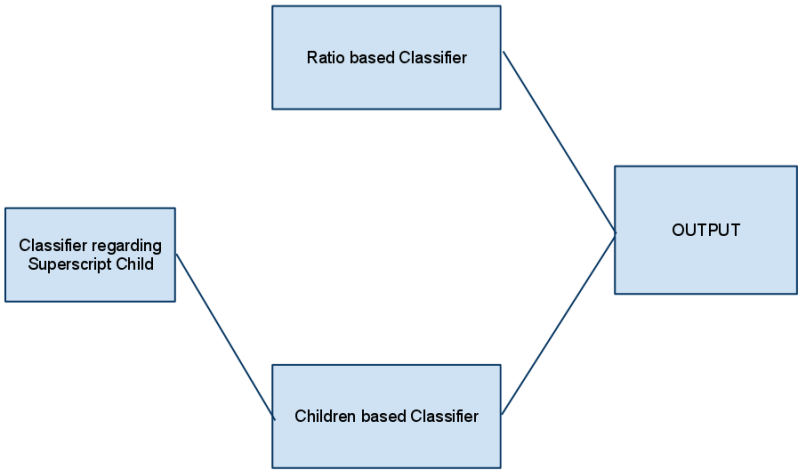

Figure 4.12: Context-Adapted Symbol Classifier

In this approach, the symbol is classified regardless of the amount of context. In the previous example, the result of the children classifier only considers the superscript based classification. It amounts to saying that the other child based classifiers would have yielded the same result as the superscript based one.

However, the missing context corresponds to a lack of information. This should be reflected in the final confidence values. Therefore, we decided that the corresponding classifiers should return a 'full-uncertainty' result, an uniform distribution of confidence over the classes. It has a smoothing effect when the results are merged.

Parameters and Training of the Components

The ratio-based classifier is a simple Bayesian inference. However, the usual Bayesian inference takes into account the prior probability of each class. It is usually computed by calculating the frequency of each class in the data set. In our data set we have more

Table 4.5: Parameters of the Symbol Classifiers

Classifier	Parameters	Number of examples	Accuracy
Parent-based	Learning rate: 0.3 Momentum: 0.2 Epochs: 500	2981	72.2%
Inline child based	Learning rate: 0.3 Momentum: 0.2 Epochs: 500	1553	70%
Other child based	Learning rate: 0.3 Momentum: 0.2 Epochs: 500	About 800	About 98%

ascending symbols, but it does not mean that ascending symbols occur more often in equations. We modified the inference by giving each class the same prior, so we have:

$$P(class \mid ratio) = \frac{P(ratio \mid class)}{P(ratio)} \times \alpha$$

where α is a normalizing constant.

This classifier is built and trained with Weka. It only classifies well 251 out of 372 examples (67.5%). It can be due to two facts. As we saw in the analysis of data, there are severe overlaps in the distribution of ratios for each class. Besides, this training is done considering the prior on symbol class, which will be removed in a later step, within the classification algorithm.

The neural networks are trained with a 10-fold cross validation. The parameters are summarized in Table 4.5. The percentage of well-classified examples is also shown. They all have 7 inputs (3 numerical for parameters (H-, D- and V-like)[2] and 4 binary for the class of the parent or child ($PCLASS$, $LCLASS$, and so on)).

A low accuracy is not always a problem, because (a) the classifiers will be associated, and some mistakes will be corrected by the other classifiers, and (b) the confidence values are more important than the classification in this context.

For the final classification, the outputs of each classifier are mixed. The confidence values are weighted sums of the different outputs.

$$\text{Final} = \alpha\times \text{ Ratio-based} + \beta\times \text{ Parent-based} + \gamma\times \text{ Children-based}$$

The children-based classifier is the most important, because it contains the most interesting parts of the context. It is given the highest weight ($\gamma = 0.55$). The parent is a different part of the context, because the relative position with respect to the parent is for example more due to the parent class than to the symbol class. It is then given a smaller weight ($\beta = 0.10$). Therefore, is really has an effect when the confidence of this classifier is very high. The ratio is a direct consequence of the class, but as we saw, is often not sufficient for a good classification. The weight is accordingly moderate ($\alpha = 0.35$). The children-based classifier's output is the average of the outputs of all child-based classifiers, because there is no reason to give an advantage to one particular relationship.

[2]for the definition of these parameters, see page 58

4.2.2.2 Relationship Classifier

The aim of this classifier is to determine, given two symbols, which is the most likely relationship between them. Similarly to the symbol classifier, we return confidence values for each class rather than a crisp classification. However, we aim at higher confidence than for the symbol classification, because the relationship classifier is also used for the parenting.

The relationship classification assumes that we know the class of the symbols involved, even if this information is wrong. It uses this class information, along with the relative size and position of the symbol to deduce the relationship. The components of the classifier are a neural network, fuzzy regions and fuzzy baselines. We will present the general form of this part of the system, describe the different components, and explain the choice of the parameters.

Form of the Classifier

The classifier is made of several independent parts, each of which giving confidence values on the relationships. These results are merged to give the final answer, which can then be compared to the thresholds. The system is presented on Figure 4.13.

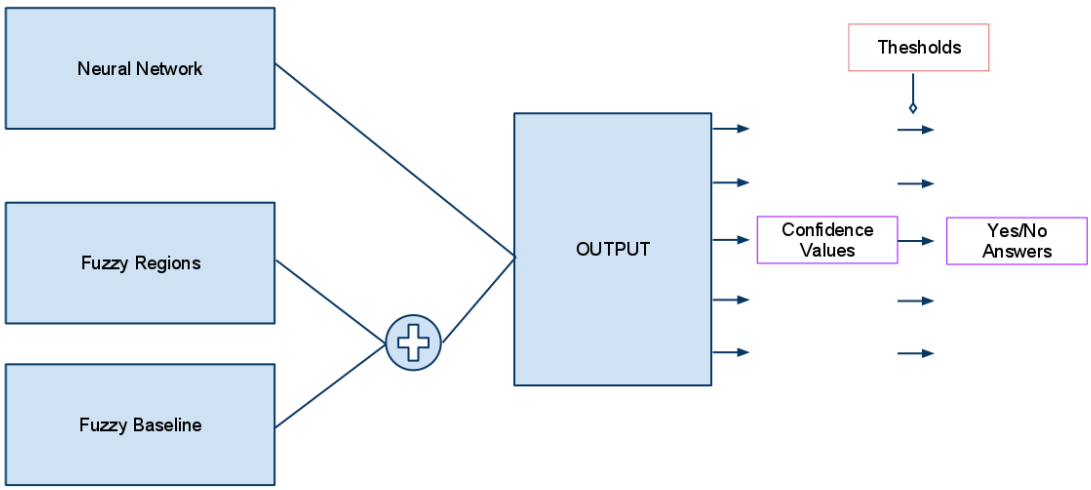

Figure 4.13: The Relationship Classifier

A Neural Network

The central part is a neural network. It has been trained by the data set to efficiently recognize the relationship between two given symbols. The inputs are parameters inspired by [25, 2]. These are:

- H - the relative size of the child to its parent: $H = \frac{h_p}{h_c}$
- D - the relative vertical position of the child: $D = \frac{y_p - y_c}{h_p}$

- V - the relative horizontal position of the child: $V = \frac{xmax_p - xmin_c}{w_p}$

where the indices p and c stand for 'parent' and 'child', h is the height, w is the width, y is the vertical centre of the bounding box, and $xmin$ and $xmax$ are the left and right bounds.

V is not used by Aly et al., and has been defined for this project, to facilitate the differentiation of subscript and under, and of superscript and upper. The idea is that, while H and D can be very similar for the 'subscript' and 'under' relationships, V can make the difference, being positive for 'under' and negative for 'subscript'.

These parameters have been proved by Aly et al. [2] to be efficient. However, they use normalized bounding boxes to calculate them. To do so, they have to know the identity or class of the symbol, and they add a virtual ascender (to the 'small' and 'descending' symbols) and a virtual descender (to the 'small' and 'ascending' symbols). We do not want to modify the bounding boxes, and we want to keep as much uncertainty as possible. Instead of normalizing bounding boxes, we also set the parent and child classes as inputs of the network.

Fuzzy Regions

The use of fuzzy regions for the structure recognition was influenced by the work of Zhang et al. in [29]. The aim of this system is to assist the neural network to classify the relationships, but also to tell when the symbols are not in a relationship, which the former network cannot do. It is particularly useful for the parenting task. The fuzzy regions are used for all relationships except 'inline', where fuzzy baselines are used instead. They consist of a crisp region, and a fuzzy area. Examples of how fuzzy regions are build can be seen on Figure 4.14.

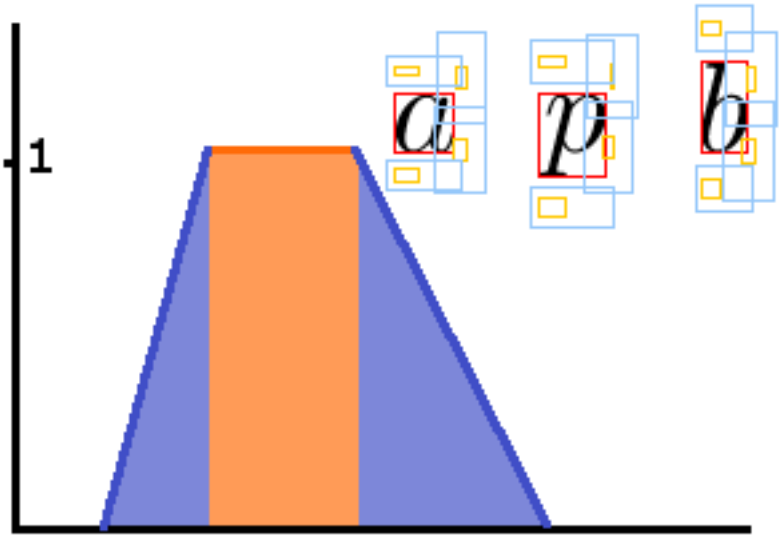

Figure 4.14: The Fuzzy Regions

To evaluate the confidence for a child symbol to be in a particular relationship with a possible parent symbol, we compute the membership value for the center of the left bound of the child in the corresponding parent's fuzzy region, as we can see on Figure 4.15.

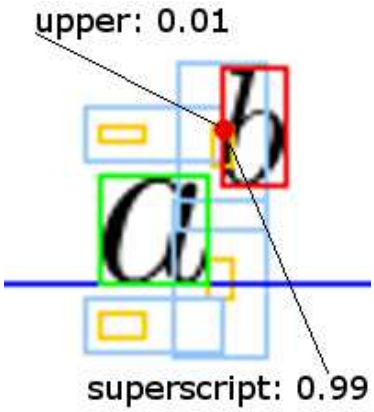

Figure 4.15: Membership Values Computation

Fuzzy Baselines

Unlike the other children, an inline child is not necessarily close to its parent. It makes the use of regions more difficult. However, by definition, an inline child is on the same line as its parent. Baselines are the lines where symbols are written. The position of the baseline of a symbol mainly depends on its class, as shown on Figure 4.16. This relative position could be learnt if we had a data set with this information. Because of the limitations on time, we decided to calculate it by hand. In fact, our project focuses on printed Latex formulae, where the position of the baseline is the same for all symbols of the same class. These positions are presented in Table 4.6, expressed relatively to the symbol's height.

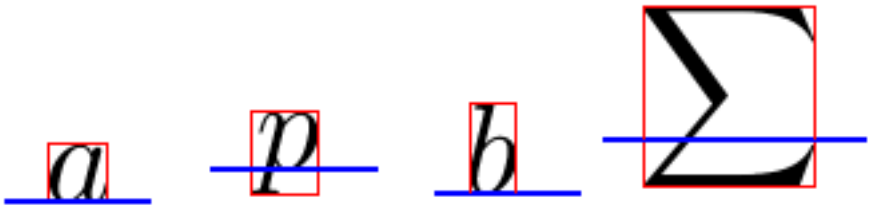

Figure 4.16: Position of the Baseline

Because we produce a flexible solution, we handle the possible variation of the writing line by considering fuzzy baselines. For a pair parent/child of symbols, we proceed as follows. First, we calculate the baseline of the parent, given its class. Then, we look at the possible baselines of the child, and calculate a confidence score based on the distance from the parent baseline and on the confidence of the child class.

Table 4.6: Baseline Positions

Class	% of the symbol's height
small	100
descending	70
ascending	100
variable range	75

The width of the fuzzy baseline depends on the size of the symbol, but the general shape is presented on Figure 4.17. It shows the confidence as a function of the distance between the baselines compared, in pixels.

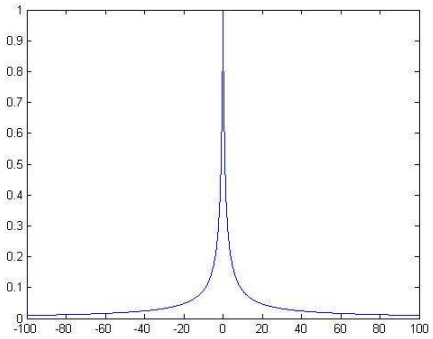

Figure 4.17: Baseline Score

Use of the Relationship Classifier

The relationship classifier is used for the identification of relationships, but also for the parenting task. The classification does not involve a simple mixing of the different classifiers, as in the symbol classification. The algorithm for this task will be described later. For the parenting task, an extra classifier is defined. The possible-child classifier, which, given two symbols, tells if the second can be a non-inline child of the first one. This decision tree takes into account the relative size and position of these symbols, as well as their classes.

When we try to find the relationship of a symbol, we first check if it can be on the same line as another symbol. We combine the confidence value given by the neural network, the baseline score and the confidence of not being a non-inline child, considering each possible parent. If no inline parent is found, we check if the symbol can be the child of another symbol, using the corresponding confidence values for the neural network, the fuzzy regions and the possible-child classifier.

Parameters and Training of the Components

Relationship Classifier

To train the neural network, we only consider the examples in the data set which correspond to symbols having a parent. The neural network has actually 11 inputs. Three are

numerical (H, D and V), and eight are binary ($PCLASS = 1 \ldots 4$, $CLASS = 1 \ldots 4$). We have 8 neurons in the hidden layer, and 5 outputs, one for each relationship. For the training, the learning rate is 0.2, the momentum is 0.2 and the number of epochs is 500. In the training set, there are ten times more inline, superscript and subscript relationships than upper and under. Thus, we used a cost matrix to train the network, such that there is no preference in the final classification for the first three relationships. 2977 of the 2981 training examples are correctly classified by the final network, but we have to remember that this network cannot tell if two symbols are not in a relationship.

Possible-Child Classifier

The possible-child classifier is a decision tree. It is trained with 3875 pairs of symbols, either inline or not in a direct relationship, and 3463 pairs of symbols in one of the four non-inline relationships, using the J48 algorithm. It has been trained with a minimum number of objects per leaf set to 30. The training with a 10-fold cross-validation reached 95.9% of correctly classified instances.

Fuzzy Baselines

The fuzzy baselines are built by hand from the observed relative position of the baseline for each class. They are used to calculate a baseline score, which have two levels of fuzziness. The first one is based on the distance of the baseline to the parent's baseline. The score is computed as follows:

$$b_1 = \frac{\alpha}{\alpha + d}$$

where d is the distance and α is a parameter based on the height of the symbol ($\alpha = 16\% \times H$). Indeed, if the fuzziness was independent from the size of the symbol, small symbols would have a higher risk to be misclassified as inline.

The second level of fuzziness is the uncertainty on the symbol class. The score b_1 is multiplied by the confidence on the symbol class to obtain the final baseline score.

Fuzzy Regions

The fuzzy regions are defined according to some statistics regarding the data set. For each class, and each relationships, we retrieve the mean and standard deviation of the parameters D and V. The means allow to define the center of the crisp regions, and the standard deviations their width and height. If $\overline{X}$ and σ_X are the mean and the standard deviation of X, given the definitions:

$$D = \frac{y_p - y_c}{h_p}$$

$$V = \frac{xmax_p - xmin_c}{w_p}$$

we define the coordinates of the crisp region:

$$\begin{aligned} x_{min} &= x - w * (\overline{V} + \sigma_V) \\ x_{max} &= x - w * (\overline{V} - \sigma_V) \\ y_{min} &= m - h * (\overline{D} + \sigma_D) \\ y_{max} &= m - h * (\overline{D} - \sigma_D) \end{aligned}$$

where x is the right bound of the symbol, w and h its width and height, and m its vertical center. The margins, or fuzzy areas, are computed using the relative size of the symbols, represented by the parameter H. The vertical (top and bottom) and horizontal (left and right) margins are calculated as follows:

$$\begin{aligned} h_{marg} &= \frac{1}{2}(\frac{h}{H_{min}} - h_{crisp}) \\ v_{marg} &= \frac{1}{2}(\frac{h}{H} - w_{crisp}) \end{aligned}$$

where H is the parameter $\frac{h_p}{h_c}$, h and w are the height and width of the symbol, and h_{crisp} and w_{crisp} those of the crisp regions. This is summarized on Figure 4.18.

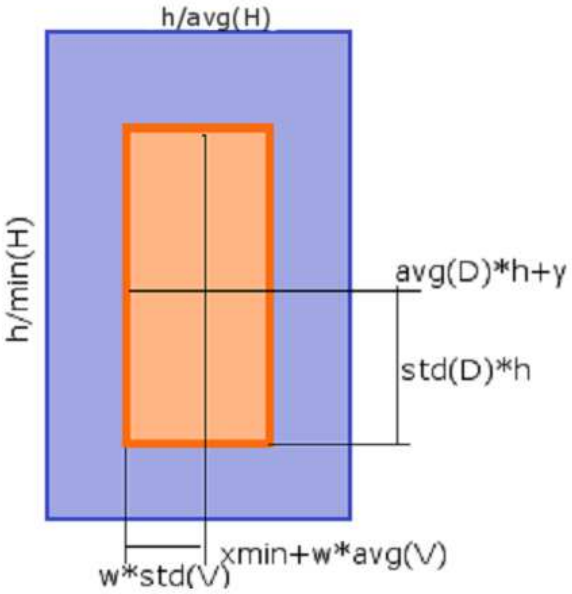

Figure 4.18: Building the Fuzzy Regions

For the relationships upper and under, the calculations are different, because the training set does not represent the complex reality. Indeed, in the training set, there is always at most one symbol above and one under. Unlike superscripts and subscripts, the horizontal positions of these children change when there is more than one symbol. The regions' vertical bounds are still defined as before. The horizontal bounds are:

- Left bound: for the crisp region identical to the left bound of the symbol's bounding box ; for the fuzzy region, the left bound of the symbol minus 15% of the symbol width
- Right bound: for the crisp region, calculated as before, for the fuzzy region, add 15% of the symbol's width

Finally, the training set does not have upper and under relationships for small, descending and ascending symbols, nor subscripts and superscripts for variable range symbols.

Therefore, we build the corresponding regions with the parameters relative to other symbol classes. For example, the parameters of variable range symbols are used to build the fuzzy region 'upper' of a small symbol.

Associating the Classifiers

When the algorithm parents symbols and labels relationships, the different classifiers are used, and a weighted sum of the values they return is calculated. A different treatment is applied depending on whether the relationship is inline or not. The weighted sum is then compared to a threshold. In this project, we found that 0.55 was an acceptable threshold, i.e. giving good results.

The coefficients for the inline relationship are

- Relationship Classifier: 1, low because it cannot tell whether the considered symbols are in a relationship
- Possible-Child Classifier: 1, low because the answer 'No' of this classifier can mean either inline or no relationship
- Fuzzy Regions: 0, because there is no fuzzy region for this relationship
- Fuzzy Baseline: 4

It means that a low baseline score can be compensated by a high certainty of the other classifiers, but the other classifiers cannot alone classify an inline relationship. The weighted sum is divided by 1+1+4=6 to be compared to the threshold.

The coefficients for the other relationships are

- Relationship Classifier: 1 (for the corresponding output) for the same reasons as before.
- Possible-Child Classifier: 2, higher because the answer 'Yes' means that the pair of symbols are in a non-inline relationship
- Fuzzy Regions: 4
- Fuzzy Baseline: 0

The fuzzy regions are selective, but a low confidence value can be compensated by a high confidence of the other classifiers.

4.2.3 Functionalities of the System

We do not only aim at a mathematical formula recognizer. As designer of this intelligent system, we want to be able to test it. This adds some functionalities to the system. In this part, we will present briefly some of these additional functionalities, that we felt important for the system to have.

The list of these functionalities is:

- Manually selecting of removing the parent of a symbol
- Manually labelling symbols and relationships
- Classifying symbols only
- Classifying relationships only
- Viewing possible relationships between any two symbols
- Handling several expressions
- Exporting the interpretations to create a test set
- Comparing the recognition with the actual interpretation

Manual selection and removal of symbol's parent

It can be useful to parent symbols by hand. For instance, when we create a test set, we need to link symbols together in order to label the relationship. When we want to test the relationship classifier independently from the parenting algorithm, it is also useful to parent symbols by hand. Implementing the possibility of clicking on a symbol and change its properties intuitively is a key part of the GUI.

Manual labelling of symbols and relationship

The other feature for the design of a test set is the possibility of manually label symbol classes and relationships. When a symbol is selected, we should have the option of selecting its class. If it has a defined parent, we can also select the relationship. By doing so, the context of the symbol is updated. The data structure used is the same as the one used by the classifier. Thus, we can for example label relationships and test the symbol classifier only.

Classification of the symbols only or the relationships only

The system is complex, and made of several classifiers. Our classification algorithm tries to recognize the expression through an iterative process, where the symbol classification and the structure recognition constrain each other. It is important to be able to test each part independently. Making the classification of the relationships only (or of the symbols only) possible allows to assume that the symbol classification and the parenting (or the context of the symbol) is known, and to test how a particular part of the system performs given this information.

Relationship table

Testing the parenting only is more difficult. Indeed, in our algorithm, the parenting is done at the same time as the relationship classification (see next section 'Presentation of the Algorithms'). More particularly, we parent symbols according to scores that are based on the relationship classification. To analyse the parenting, understand errors, and adjust scores thresholds, we need to display a 'relationship table'. To build it, we use the fact that in most cases, the parent is on the left of its children. For each symbol, we classify the relationship it might have with every symbol which is on its right side. An example of this table is shown on Figure 4.19.

Figure 4.19: Relationship Table

The table is only built for analysing and designing the system. In the actual parenting algorithm, we do not need such a complexity ($O(N^2)$ classifications, if N is the number of symbols).

Processing several expressions at the same time

Opening an input file, processing it, analysing the results and closing it is a long operation, especially when we want to handle a test set. In this case, it is also interesting to see the performance on the whole set rather than on each instance in the set individually. Therefore, we allow the system to open several files in once, or to read several expressions in a Latex file. In the workspace, we then have a list of expressions rather than only one. The particular usefulness for the test set building is explained in the next paragraph.

Creation of a test set

The system is built by training each classifier individually. We need a big data set, which was built automatically. This process cannot be applied to the creation of a test set. Indeed, we have to test the whole system. In particular, the parenting cannot be tested with a database composed of symbol instances. Therefore, we implemented the possibility to let the designer build a test set. To do so, we can open several input expressions,

manually parent the symbols and classify them and their relationships. Then, the test set should be stored as an XML file, and loaded again later to compare the recognition of the expressions with it.

Testing the system

To evaluate how good the system is, we have to test is. That means, we have to compare its results with expected results, which are stored in a test set. To test the system, we open a set of expressions, the same as the one used to create the test set. Then, we ask the system to recognize these expressions. We obtain the symbols classes, parents and relationships. We then read the XML test set. We can finally compare the lists of symbols in the recognized expression and in the expression from the test set.

The comparison of two expressions returns a set of performance features. It includes the following information:

- Percentage of errors on symbol class
- Percentage of errors on relationship class
- Percentage of parenting error (when symbols are associated with the wrong parent)
- Symbol correctness score
- Relationship correctness score

The correctness score illustrate how confident the system is on the right class or relationship. These scores are explained in the chapter 'Results and Evaluation'.

4.2.4 Integration of the Classifiers in an User-Friendly Interface

It is good to have a code which can perform many actions. However, it is a waste of time if we have to re-launch the program for every action we want to perform, and every time we change the input. Therefore, we integrated the system in a graphical interface. It is a benefit in time for two reasons. First, we have buttons that we can click for each action, and in particular we can test several inputs by opening several files without closing and starting the program again. Second, when something does not work, we can spot the origin of that dysfunction by looking at all the information displayed.

We include in this 'user-friendly' part of the system a set of functionalities, and a set of features. To have more details, report to the 'Implementation' chapter.

4.2.4.1 Functionalities of the Interface

Besides providing buttons and menus to execute the functionalities described before, the interface should have two main functionalities:

- Display all relevant information, as well as the recognition results, and
- Allow an easy setting of inputs.

Display information and results

The goal is to give the user and the designer of the system the important information about the expression and the recognition. The display should be organised, intuitive, and easy to use. This is described in more details in the following part - 'Features of the interface'.

Set inputs

Another important function is to make the setting of inputs quick and easy. Due to our requirements, we should be able to load one or several inputs. We have implemented four ways of setting inputs:

- By loading binary images,
- By entering a LaTeXcommand,
- By reading a LaTeXfile,
- By reading an XML file.

When we click on an image file in the workspace, we can choose to open the image. We can also click on an 'Open' menu. We have designed a window with a text field, where the user can type a Latex formula. It is then automatically converted into an image. We have also implemented a Latex parser. It reads a Latex file, and retrieve the text between the tags '`begin{equation}`' and '`end{equation}`'. Each one of them is then converted into an image. For each of these methods, the images are then segmented, and the representations of symbols and expressions are created before it is included in the interface, to be used.

The last possibility is to read an XML file. The document should be in the form presented in the section about data representation, in the paragraph 'Exporting the results'. In the document, the bounding box information allows us to display the layout of the expression. The XML file should also contain the interpretation of the formula.

4.2.4.2 Features of the Interface

An organized visual display of the information is often easier to understand than a text output in a console. The aim of an interface is to provide this display. The set of features can be divided into five major parts:

- Display of the input
- Display of the input's properties
- Display of the classifiers outputs
- Display of the results
- Display of the file system

We will present these features and why they are interesting.

Display of the input

The input display allows to see the image input - the mathematical expression - and its segmentation. We can check if the segmentation is right. It shows the bounding boxes of symbols. We should also have the possibility of clicking on a symbol to display its information (e.g. width, height, class) and the result of its classification.

We should also be able to show all the visual information about this expression. For example, we can display the baselines with their confidence, the links between symbols, corresponding to the parenting, or the regions associated with the symbols.

Finally, we want to be able to process several inputs at the same time. Thus, the interface contains several layers, one for each expression. Menus are provided to jump from one input to the other.

Display of the input's properties

When we click on a symbol, it would be interesting to see its features, so we can for example understand the input of the classifiers and the possible mistakes.

Display of the classifiers outputs

Each part of the multi-classifier system, presented before, returns confidence values on the symbol class and relationship with the parent. The outputs are merged to give the final result. It is interesting to show the output of every part, to see how they perform. For each part, the output is presented in the form of an histogram, showing the confidence value for each class.

Display of the results

When we have recognized an expression, it is interesting to show the results, in terms of the goodness of the interpretation. It is used in two situations. For the mere recognition of an input, we show the global confidence on the symbol recognition, on the relationship recognition, and of the whole recognition. When we test the system, that is when we compare the actual interpretations with the expected ones, we show a correctness score, or accuracy.

For the latter case, we plot the results. We can choose which score to display (e.g. symbol correctness score as a function of the number of symbols). It gives more information than displaying simple text. We also have a window to output the results in a textual form, to show the exact figures.

Display of the file system

The use of the system involves opening input images, loading classifiers, reading tex files, or importing XML interpretations. To make that process easier and faster, we set a sort of workspace. A window displays the file system of the computer, highlights files that can be used by the program, and shows the possible actions available when a file is clicked.

4.3 Presentation of the Algorithms

We cannot present all algorithms used in the project, but the most interesting ones can be found in the appendix 'Algorithms'. In this section, we will present three algorithms. The first one is the first of the process - the segmentation algorithm. Then, the most important algorithm, used for the recognition, is explained. Finally, we show how we can export the results, or manually labelled interpretation, to create a test set for example.

4.3.1 Image Segmentation

For the recognition, we assume that we know the bounding boxes of the symbols. These are the inputs of our system. To build a data set, or to construct a test set for the system, it is more convenient to have a segmentation algorithm, so that the inputs are calculated directly from images.

There are very efficient segmentation algorithms existing, but we chose a very simple one. We make the strong assumption that all printed symbols are connected. This is not the case for i, j, $=$, and so on, but we will ignore them for the sake of simplicity. The chosen algorithm is a two-pass algorithm for the detection of connected components.

The Algorithm

Two-pass means that we go only twice through the image. The idea is to label the connected components present in a binary image, with a different label for each component.

In the first pass, we label the image, and we record the equivalences between labels. To do so, we proceed as follow, for each pixel:

1. if it is a background pixel, move to the next pixel
2. otherwise, look at the labels of the surrounding (already labelled) pixels
3. if there is no label around, mark the pixel with a new label
4. otherwise, mark the pixel with the lowest label, and record the equivalence with the adjacent labels.

At the end of the first pass, every foreground pixel has a label. Several pixels in the same component may have a different label, but they are all equivalent to the lowest label in that component.

In the second pass, we update every pixel's label with the equivalent label. Normally, after this pass, there is only one label in each component. The goal is to retrieve the bounding boxes, so during the second pass, we also record the minimum and maximum coordinates in both x and y directions. The algorithm returns a list of bounding boxes. We can see how the algorithm works on Figure 4.20.

4.3.2 The Recognition

The classification algorithm, or recognition algorithm, is the most important of the system. It consists of two parts. First, the symbol classification only creates the neural networks inputs and merge their outputs, for each symbol. Then, the structure recognition

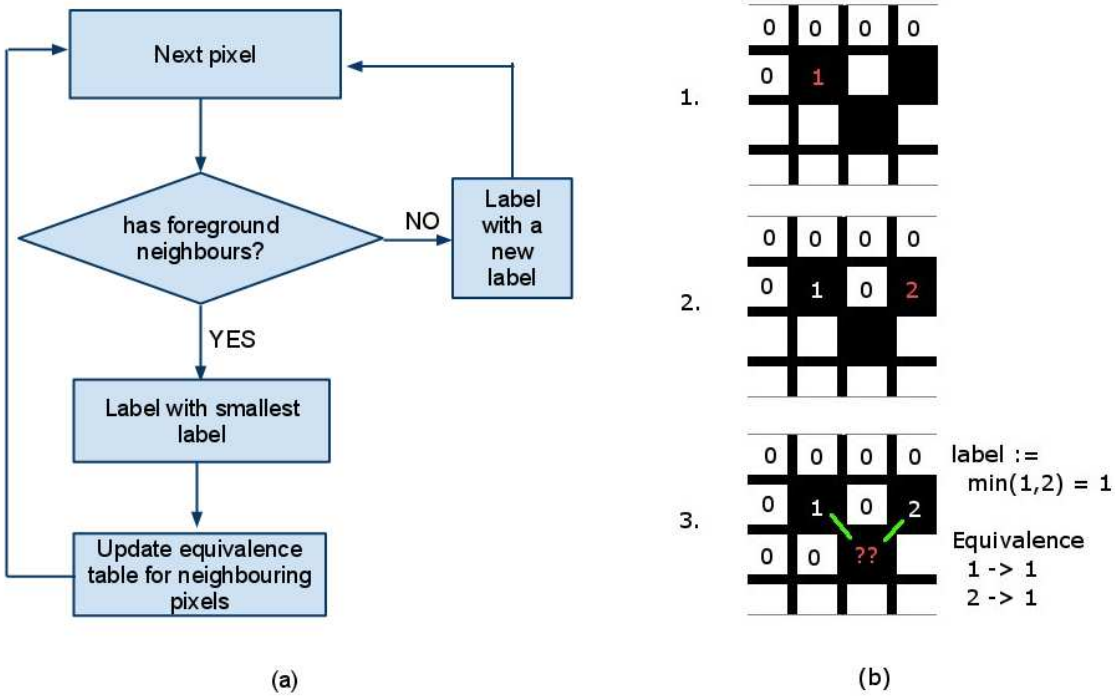

Figure 4.20: Segmentation Algorithm

algorithm is more complicated, because it computes two tasks. If finds the parent of each symbol, and the relationship between them. In the following, we will describe the different steps of the recognition.

4.3.2.1 Rough Symbol Classification

The first step is the classification of every symbol based on the ratio width/height. The Bayesian Network returns confidence values, which are divided by the prior on symbol classes, to obtain the scores described previously. The algorithm is summarized on Figure 4.21.

4.3.2.2 Structure Recognition

We tried to design a fast structure recognition based on machine learning techniques. We wanted to limit the backtracking and the recursion for the parenting task and the identification of relationship. Comparing each possible pair of symbols is out of the question. Our algorithm does the parenting in one pass, with a limited backtracking.

Inline relationships are more difficult to find because the symbols involved can be far apart. Other relationships involve symbols close to each other. Therefore, we first try to see if a symbol is inline with another before checking for other relationships. Moreover, a symbol can be inline with several others, whereas it can only be the subscript of one.

Given one symbol, we first check whether it is an inline child. We compute the score, as explained before, considering only the last symbol of each baseline as possible parents. The baselines are considered from the most dominant to the most nested. If one score

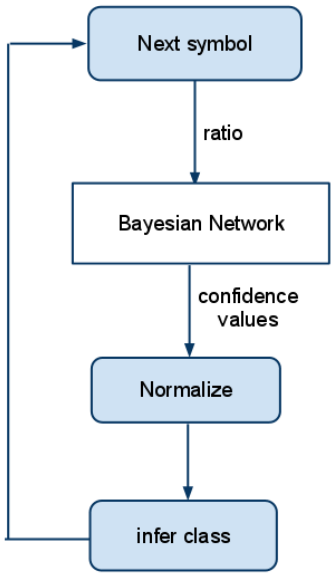

Figure 4.21: Rough Symbol Classification

is higher than the threshold, we stop the search and do the parenting. The list of last symbols for each baseline is updated.

If no inline parent is found, we browse the stack of last symbols seen, and we check whether the considered symbol can be a child of one of these last symbols. Since a non-inline child is often close to its parent, the parent is likely not to be far in the stack. Hence the backtracking is limited. We also remember the most likely parent and relationship. Thus, if no confidence is higher than the threshold, we check if there is no child for the most likely parent in the most likely relationship, and we can parent the symbol. Otherwise, the symbol is left without a parent.

The complexity of the backtracking is in the worst case $2 \times R \times C$ where:

- R is the number of non-inline relationships, which also corresponds to the maximum number of nested baselines created from one symbol
- C is the complexity, so the number of last symbols on baselines is lower than $R \times C$, as is the number of possible children for these last symbols.

The algorithm has just one pass, so the total complexity is $2RCN$ in the worst case. Even if $RC > N$, not every symbol has a child for each relationship, so in practice the complexity is less than N^2. The algorithm is explained on Figure 4.22, and an example is shown on Figure 4.23.

4.3.2.3 Symbol Classification using Context

For the symbol classification using context, the inputs of the networks corresponding to each existing piece of context are computed, and the outputs of the networks are collected. The final confidence values are calculated as explained before, and the class is inferred. This algorithm is summarized on Figure 4.24.

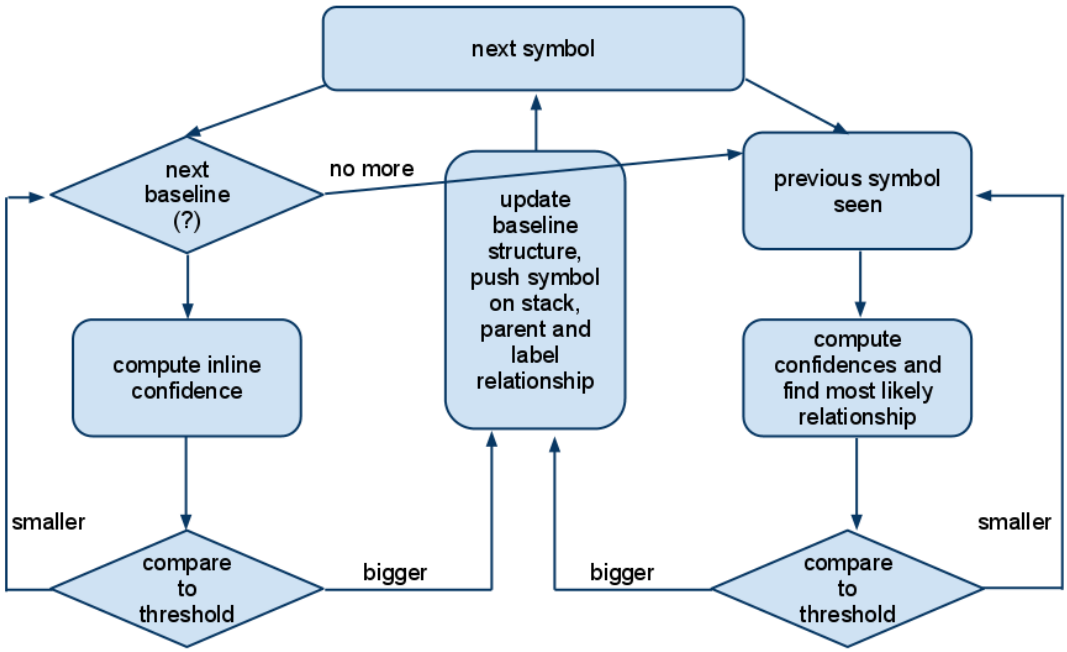

Figure 4.22: Structure Recognition Algorithm

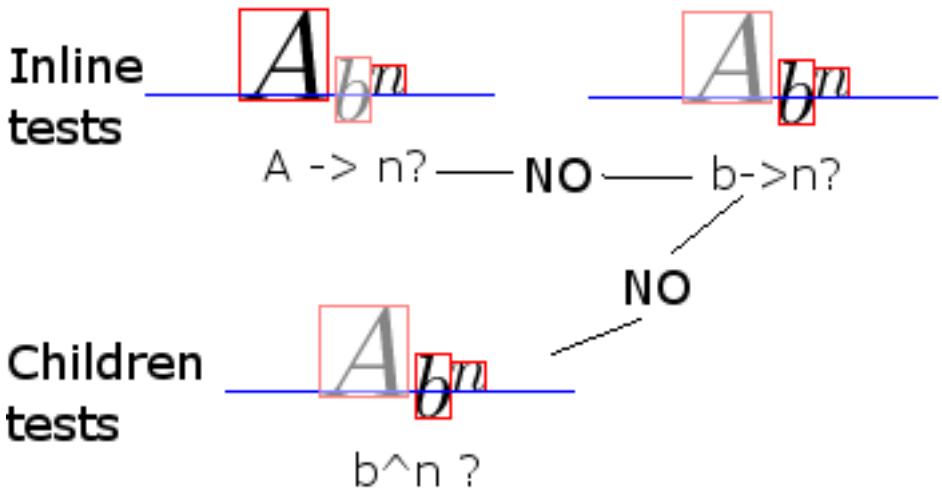

Figure 4.23: Structure Recognition Example (fon n in A_{b^n})

4.3.2.4 The Iterations

The iterative process is initialized with the rough classification. Then, at each iteration, three operations are performed. First, all existing relationships are undone. Then, the structure is recognized, and finally, the symbols are classified. This is shown on Figure 4.25.

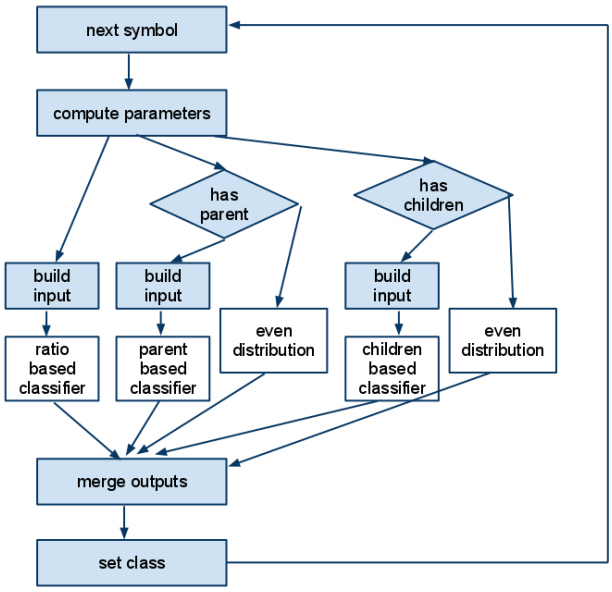

Figure 4.24: Symbol Classification Algorithm

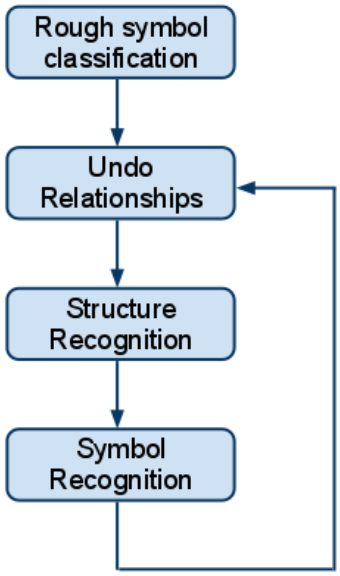

Figure 4.25: The Iterative Algorithm

4.3.3 Exporting the Results

The last step of the recognition is exporting the results. In fact, we do not perform the whole recognition, and exporting the results could allow another program to read the interpretation and perform more actions, such as the symbol recognition. Besides, it allows to save a test set, so we can evaluate the system later.

The algorithm transforms the expression tree as it is implemented in the system into an XML tree containing enough information to store the interpretation. The process is

recursive. Considering a symbol, its properties (bounding boxes, classification) are stored in nodes. The children symbol nodes are created recursively. Indeed, each child correspond to a nested expression and can be processed the same way.

Chapter 5

Implementation

The implementation is the actual realization of the designed system. It consists of the design and the writing of a program which can be run on a computer, to perform the recognition task. It should also represent the data in an efficient way, contain the different functionalities described before, and present the whole system in a user-friendly interface. The translation of these functionalities into use-case diagrams can be found in the 'Implementation' appendix.

We chose an object-oriented approach because it translates well the division of the whole system into different components. Indeed, we can identify:

- a core part, which represents the expression and perform the classification
- an input/output part to avoid redoing always the same things
- a graphical user interface (GUI)

The key part for implementing an object-oriented system is to model it. For instance, we have to divide it into different classes, each of which having a particular role. The proposed implementation uses frameworks (Weka, JDOM and JLatexMath) and a database, implemented with MS Access. The developed system is divided into several packages, presented on Figure 5.1, and explained in the appendix.

Section 5.1 explains the implementation of the expressions and the classifiers. Section 5.2 breifly presents the implementation of some additional functionalities. Section 5.3 introduces the implementation of the GUI. Section 5.4 presents the objects implementated for the structure recognition. More details can be found in Appendix A.

5.1 Implementation of the Expressions and Classifiers

The core of the system is the recognition mathematical expressions. We presented previously the representation of the data and the form of the classifiers. In this part, we will enter in more details about how this representation is implemented. We will present the different classes, their association, and their role.

5.1.1 Implementation of the Expressions

As explained before, the expressions are represented in terms of the symbols they contain, considered in their context. It corresponds the five classes of objects, presented in the

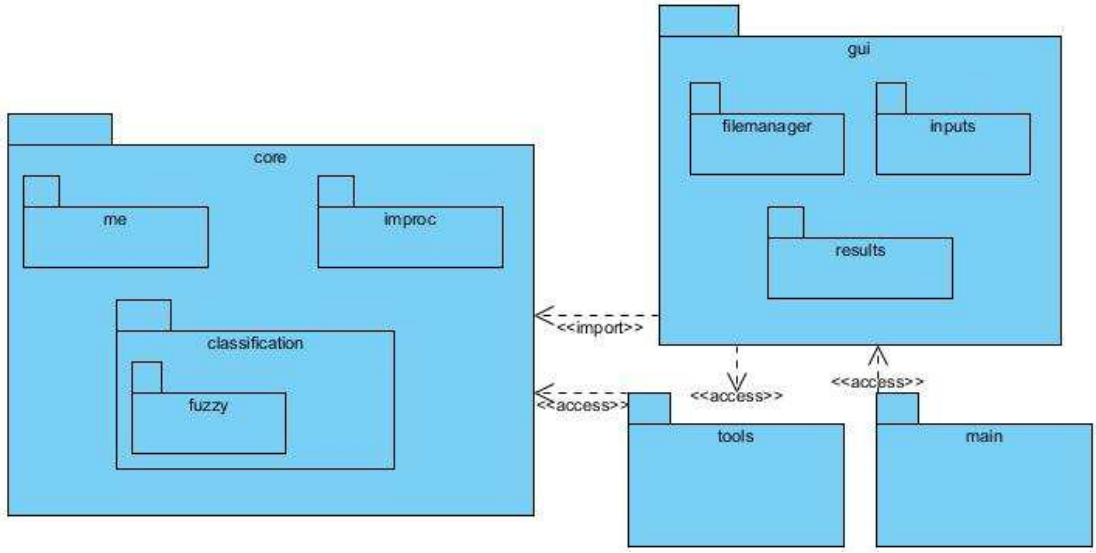

Figure 5.1: Package Diagram

class diagram on Figure 5.2. These classes constitute the content of the package 'core::me'. We represented only the most useful methods. Most of them are self-explanatory. The details are exposed in Appendix A and in the API provided on the attached disk.

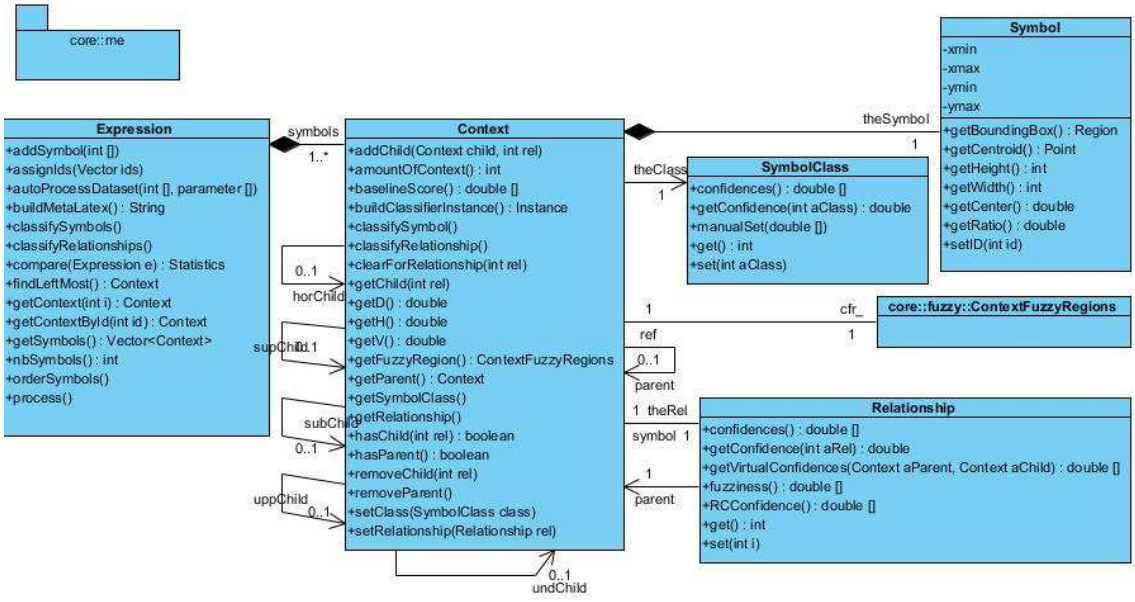

Figure 5.2: Expression Implementation

Class Symbol

The objects in this class are very basic representations of symbols. In fact, they contain information about the symbol itself, out of any context. A `Symbol` object is created from the bounding box. It provides geometrical information such as the height, the width, the position, or the centre of the bounding box. The symbol id is especially useful when testing

the system.

Class SymbolClass

A `SymbolClass` object represents the whole classification of a symbol. Therefore, it contains the output of all the classifiers involved in the classification, and the way they are mixed to yield the final confidence values.

Class Relationship

Similarly, a `Relationship` object represents the classification of the relationship existing between a parent and its child. It stores the output of the different classifiers, and the final confidences.

Class Context

An object in this class is referred to by the word 'symbol' throughout the dissertation. Indeed, our approach not only recognizes the structure, but also uses it to find the class of symbol. Therefore, the context of the symbol is as important as the symbol itself. A `Context` object is a node in the final expression tree, since we can link it to other `Context` objects. As we explained, the node also contains a `SymbolClass` and a `Relationship` object. Its basic role is to represent the result of the recognition.

Class Expression

The `Expression` object allows to see the expression more globally. It is basically a list of all symbols, sorted from the leftmost to the rightmost symbol. It provides an easy way of retrieving the information regarding the whole expression, such as its interpretation, its dimensions, the leftmost symbol (root of the tree structure after the recognition), or to apply some action to all symbols (e.g., for the symbol classification).

5.1.2 Implementation of the Classifiers

The classifiers are used to recognize the structure. First, the trained classifiers, the neural networks for example, are all stored in one independent class, whereas fuzzy regions are defined with respect to a symbol. Fuzzy baselines do not correspond to a particular class. Finally, the structure recognition is performed by the `RelationshipFinder` class.

5.1.2.1 Trained classifiers

All classifiers which need a training from the data sets are trained, saved, imported, stored and used within a single class, `Classifiers`. It adapts the Weka framework to the needs of our project. The attributes of this objects are the classifiers and the data structures needed to build an instance that can be read by the corresponding classifiers. They are summarized in Table 5.1.

This class implements three kinds of methods:

- save and read methods use the serialisation provided by Weka, to avoid training the classifiers every time the program is executed.

Table 5.1: Implementation of the classifiers

Classifier	Class	Description	Data Structure
`SCA`	`weka.classifiers.bayes.BayesNet`	Ratio-based symbol classifier	`dataStructSCA`
`SCB`	`weka.classifiers.function.MultiLayerPerceptron`	Parent-based symbol classifier	`dataStructSCB`
`SCC1..5`	`weka.classifiers.function.MultiLayerPerceptron`	Children-based symbol classifiers	`dataStructSCC1..5`
`RC`	`weka.classifiers.meta.CostSensitiveClassifier`	Relationship classifier	`dataStructRC`
`YN`	`weka.classifiers.tree.J48`	Possible not-inline classifier	`dataStructYNC`

- train methods do the training of the classifiers, with cross-validation, and build the data structures.
- classify methods, such as `getSymbolClassA( Context )`, allow the classification of a symbol or a relationship by a single classifier, and returns a distribution of confidence values.

This is summarized on Figure 5.3.

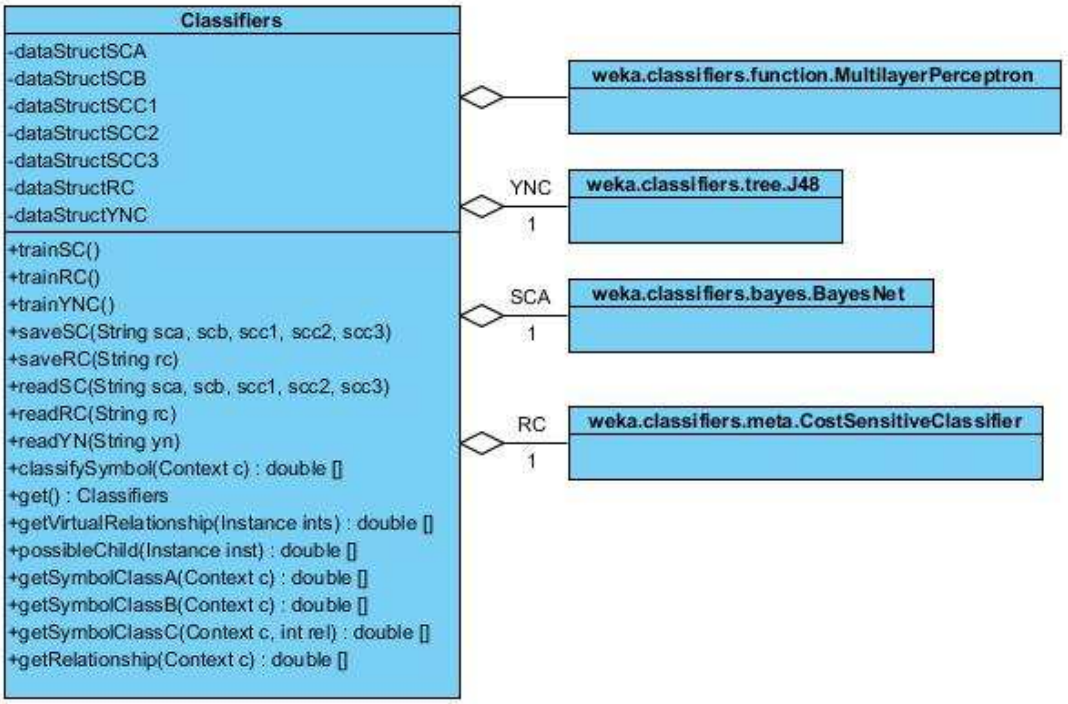

Figure 5.3: Classifiers Class

5.1.2.2 Fuzzy Regions

An overview of the classes of objects related to fuzzy regions is shown on the class diagram on Figure 5.4.

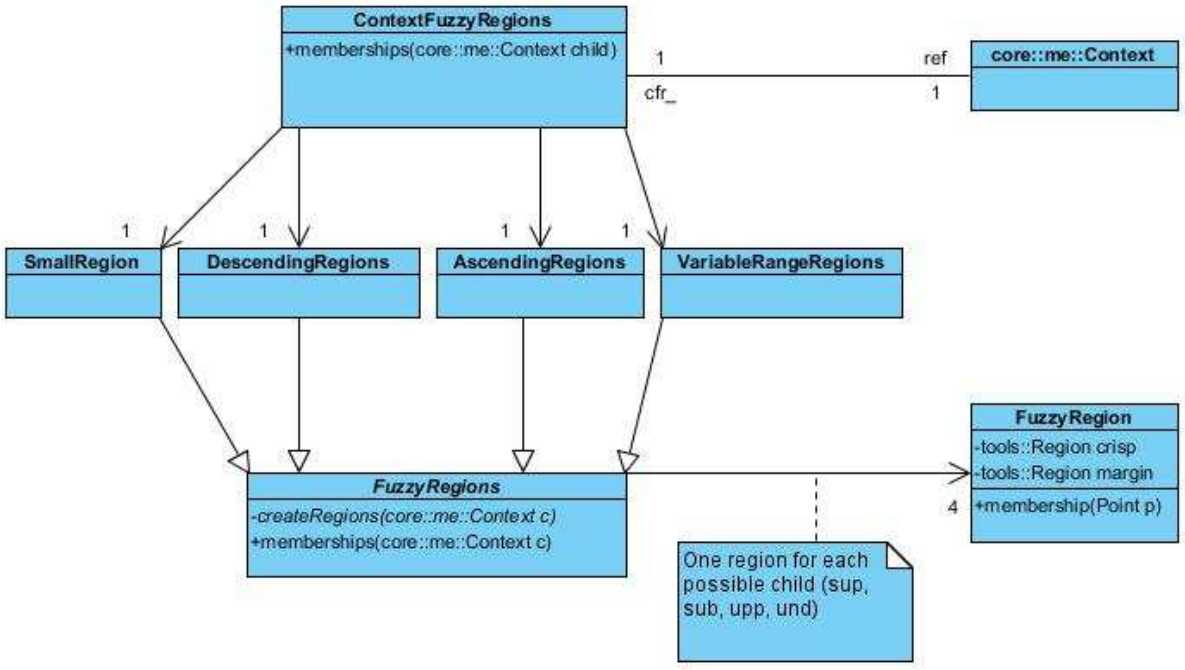

Figure 5.4: Fuzzy Regions Implementation

Each fuzzy region is represented by the class `FuzzyRegion`. The `membership()` method calculates the membership value of a point in this region. Four regions (one per possible relationship) are grouped into a `FuzzyRegions` object, to represent all fuzzy regions associated with an object. A second level of fuzziness is introduced by considering the uncertainty on the symbol class.

Each symbol is associated with a `ContextFuzzyRegions` object, which contains four sets (for the four possible symbol classes) of fuzzy regions. These are objects of the classes `SmallRegions`, `AscendingRegions` and so on. The membership values are computed for each regions and each symbol class, and then summed up, weighted with the confidence of the corresponding symbol class.

5.1.2.3 Fuzzy Baselines

The baseline scores are computed as previously explained by the `baselineScore(Context)` method in the `Context` class. The argument is the parent symbol, and the method returns an array containing the calculated baseline scores, based on the distance of the symbol's baseline to the parent baseline, for each symbol class.

5.1.2.4 Structure Recognizer

The structure recognition is performed by the object `RelationshipFinder`. It is associated with an `Expression` object, in which symbols are already classified. The method

`parentAndRelate()` implements the structure recognition algorithm. The details about the implementation of the algorithm can be found in section 5.4, and the code is presented in Appendix B.

5.1.3 Tools for the Analysis of the Classification Results

Two classes represent tools for the analysis of a classification, namely `Interpretation` and `Statistics`. The first one stores the information about the interpretation of an expression, that is, the global confidence score concerning symbol classes and relationships, the Latex equivalent of the expression, the XML document corresponding to the interpretation, and the validity of the expression.

The goal of `Statistics` is to test the system. It stores information such as the percentage of symbol class, relationship or parenting error. It is used when we compare the actual expression recognition to the expected one, saved in XML format on the computer. More details about this class can be found in the next chapter 'Results and Evaluation'.

5.2 Additional Functionalities

Most functionalities, both crucial (e.g. the symbol classification) or annex (e.g. the visualisation of the results), are implemented by the classes presented before, or by the GUI (next section). In this section, we will briefly present three classes, which provide essential tools for building and using the system: `LatexParser`, `DatasetBuilder`, and `XMLCreator`.

LatexParser

This class essentially uses the jLatexMath framework to provide tools to parse Tex files and expressions. In particular, it implements a static method, `getLatexImage(String)`, to create the input image from an equation. Moreover, the constructor of this class parses a Latex file to find the equations between the tags `\begin{equation}` and `\end{equation}`. It allows to import a set of expressions from a Latex file.

DatasetBuilder

This class provides the algorithms to generate the data sets (see Appendices A and B).

XMLCreator

This class provides the algorithms to export and import XML files, such as expressions interpretations or lists of symbols (see 'Implementation' appendix). It also allows the creation of the web page used for human labelling (see 'Results and Evaluation' chapter). It uses the JDOM framework.

5.3 Implementation of the Graphical User Interface (GUI)

The GUI is a crucial tool for building the system, the test sets, analyze the performance, and so on. Having a graphical interface makes the use of the system and the visualization of the results easier and more intuitive.

The classes that implement the GUI are in the 'gui' package, and are built with Java Swing. We can divide it into four components:

- the main window, providing the basic operations and displaying the most important results
- the results part, providing extra visual tools for the analysis of the performance
- the input part, which makes it easy to quickly input new images
- the file manager, which supports the basic input/output operations

This section will be divided into four parts, overviewing these four major components. For more details, report to Appendix A.

5.3.1 Main Window

An instance of that window is shown on Figure 5.5. We can see different features:

- The image panel, displays the expression and information relative to it
- Two menus - a top menu and a set of buttons
- A classification panel
- A property panel

These correspond to different classes, presented on the class diagram on Figure 5.6.

The aim of the image panel is to provide a view of the expression, a way to interact with its symbols and to display properties and results. It is implemented by the class `ImagePanel`. Each `ImagePanel` is associated with a `BufferedImage` object representing the input image, and to an `Expression` object to access its properties.

When a symbol is clicked on the image panel, it is selected, and its classification and properties are displayed in the corresponding panels. On this panel, we can also choose to display the baselines, the fuzzy regions, the parenting links, for all symbols or only for the selected one. In the main window, the image panels are grouped into a `MultiImagePanel` object, which is merely a list of image panels. It allows the GUI to handle several expressions at the same time. One can also click two symbols and make the parenting and relationship between them. An instance of the `SelectRelationship` window opens to select the type of relationship (see Figure 5.7).

The classification panel, instance of `ClassificationPanel` is used to display the classification of the selected symbol. The outputs of the different symbol classifiers are presented as histograms (automatically drawn with the `Histogram` class). The confidence values for the relationship classification is also shown.

The property panel, instance of `PropPanel`, shows some properties of the selected symbol, such as the coordinates of the bounding box, the symbol and parent class, and the kind of relationship. It allows to quickly check the result of the recognition.

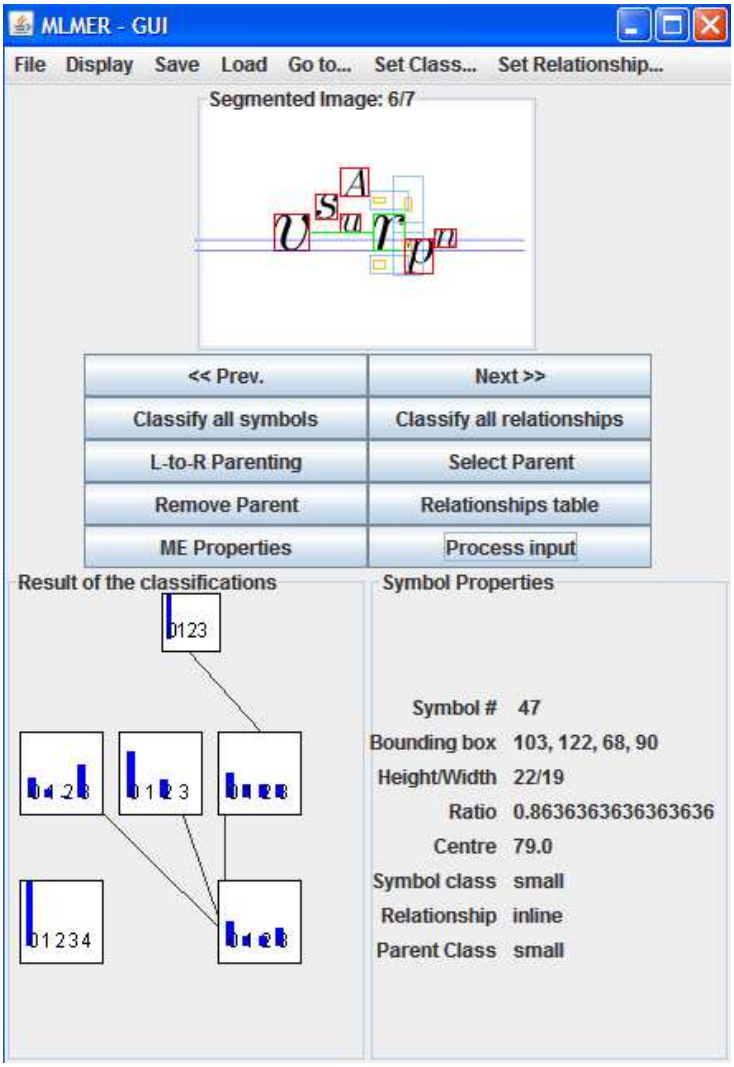

Figure 5.5: Main Window

5.3.2 Result View

Different windows can display different sorts of results:

- `ResultView` displays textual information
- `RCWindow` displays classification information
- `PlotWindow` displays results concerning the performance of the system

5.3.2.1 ResultView

The object `ResultView` is a window containing a single text area, which allow to display results and messages. An example is provided on Figure 5.8.

5.3.2.2 RCWindow

To analyse the structure recognition, in particular to understand the mistakes made, it is useful to have a look at the relationship classification for every two symbols. The

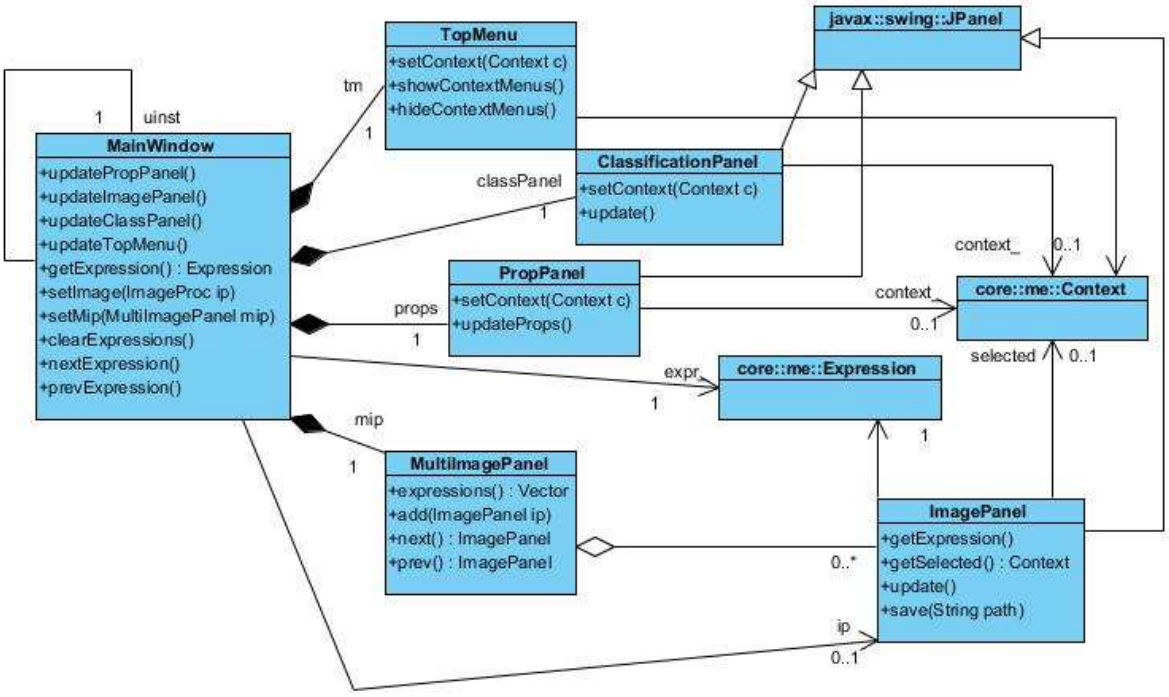

Figure 5.6: GUI Class Diagram

Figure 5.7: Selecting a Relationship

`RCWindow` object is instantiated with an expression, and the relationship classifiers (the neural network (RC), the fuzzy regions and baselines, and the possible-child classifiers) are used to display the confidence values for every pair. An example is shown on Figure 5.9.

5.3.2.3 PlotWindow

When the proposed system is tested, the performance yielded for a particular test set can be plotted in a `PlotWindow` (Figure 5.10), in order to make the analysis of the results easier. One can choose the figures to plot in a top menu.

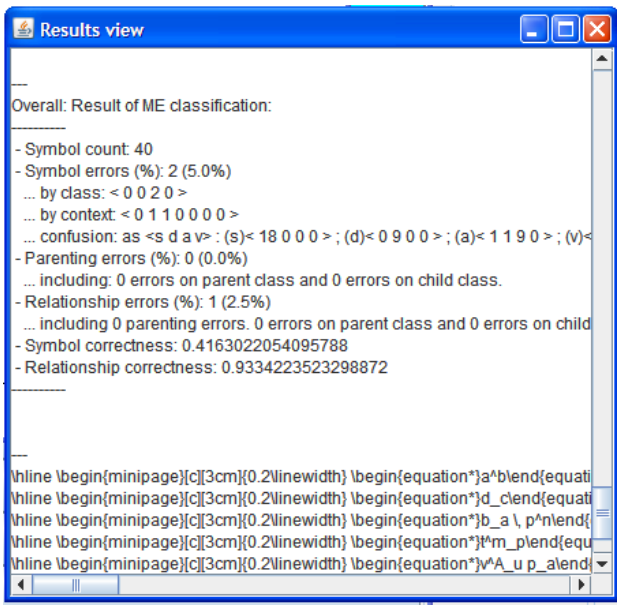

Figure 5.8: Textual Results

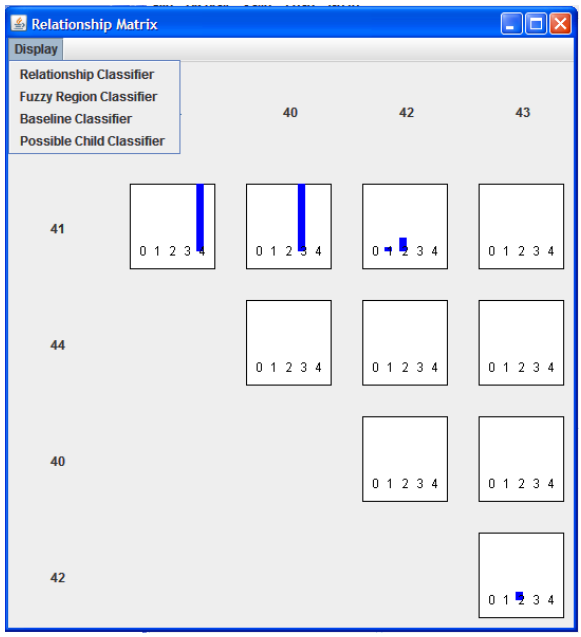

Figure 5.9: Relationship Table

5.3.3 Input Windows

Besides opening a binary image or an expression saved in an XML format, there are two windows for setting inputs directly in the GUI. The first one (class `DrawWindow`) allows the user to draw (write by hand) an expression. An instance of such a window is presented on Figure 5.11.

The second one is a `LatexToImage` window. It contain a text field for the user to type a Latex command. The class `LatexParser` is then used to transform this command into an image. An example is shown on Figure 5.12.

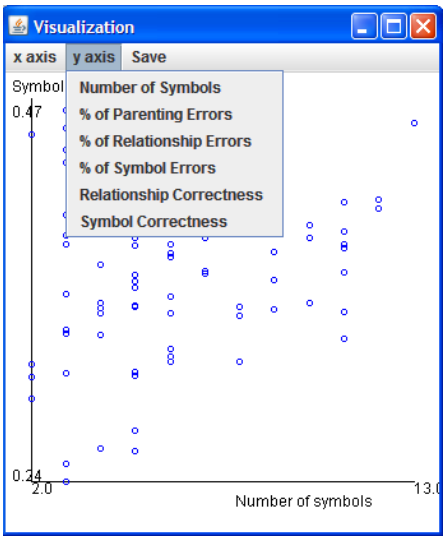

Figure 5.10: Plot Window

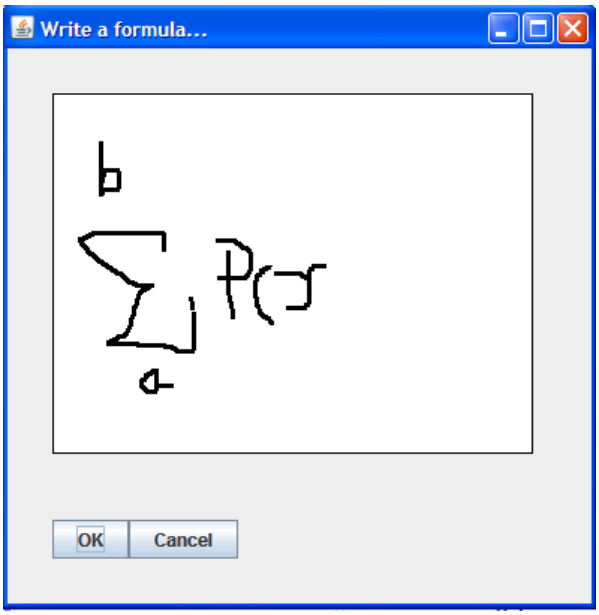

Figure 5.11: Drawing Window

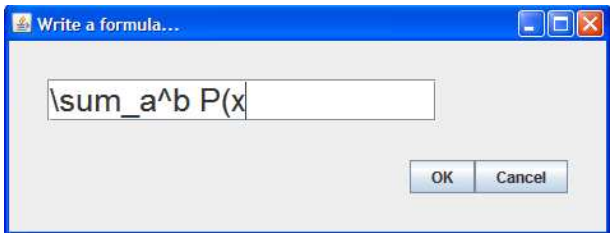

Figure 5.12: Latex Command Window

5.3.4 File Manager

Using the system involves many 'open' and 'save' operations. To make those quick and easy, we implemented a `FileManager` object. It shows the files and folders of the

current workspace. Folders, usable files (XML files, images, and so on) and other files are represented with different colours. An instance of this object is presented on Figure 5.13.

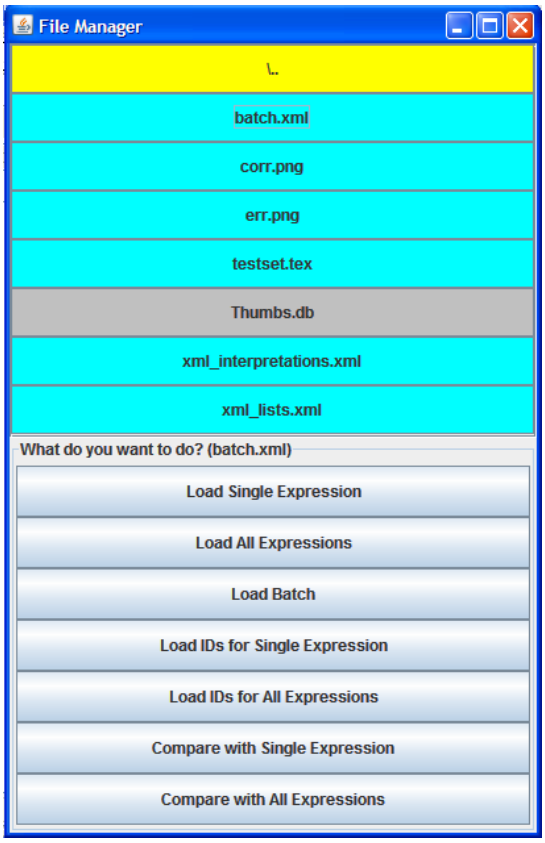

Figure 5.13: File Manager

5.4 Implementation of the Structure Recognition Algorithm

The structure recognition is performed by the method `parentAndRelate()` of a `RelationshipFinder` object. It uses the classifiers as they were described before, and two data structures. To access easily the last symbols on each baseline, we defined a class `BaselineStructure`. The last symbols seen are accessed via a stack.

5.4.1 BaselineStructure object

The purpose of the `BaselineStructure` object is to access the last symbols on each baseline, without having to browse the whole expression tree. It is made of a symbol and a list of nested baselines. Indeed, a symbol can only be a child of the last symbol of a baseline. For instance, in ab^cd, c and d cannot be children of a, but they can be children of b, which is the last symbol on its baseline.

When an inline child is found, a new `BaselineStructure` is created, with an empty list of nested baselines, and replaces the `BaselineStructure` of the parent. The whole baseline structure is simplified, and ensures that the complexity does not grow exponentially with every symbol. When a non-inline child is found, a `BaselineStructure` is created and

added to the nested baselines of its parent. Figure 5.14 shows the evolution of the main baseline structure while the successful recognition of $a^{b^{c}_{d}e}g_{h}k$

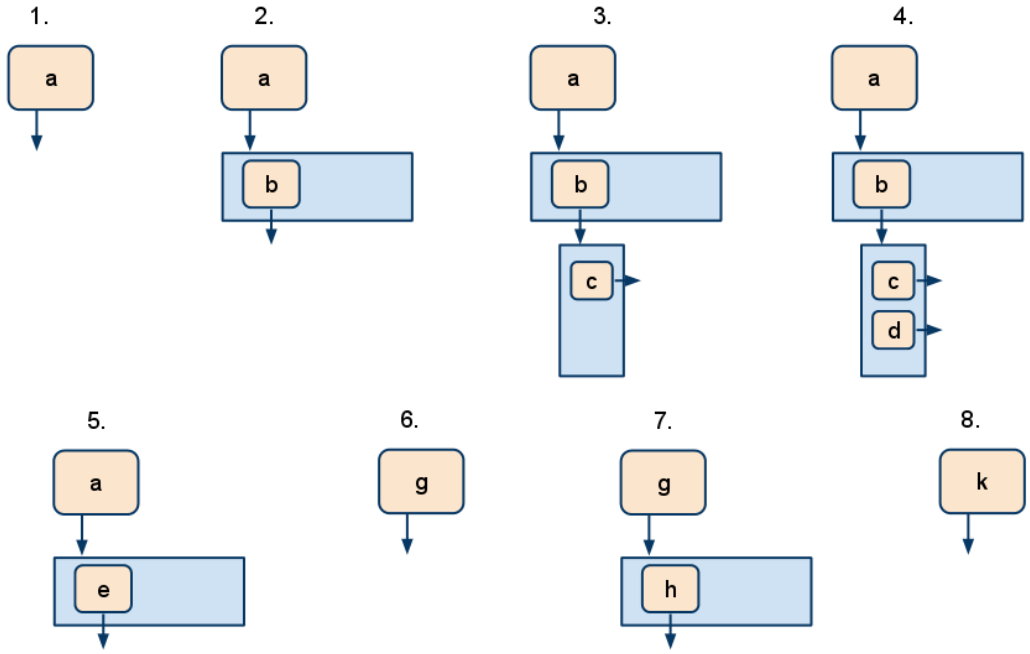

Figure 5.14: Baseline Structure

5.4.2 The Stack of Last Symbols Seen

Every time a symbol is parented, the `BaselineStructure` created is added to the stack of last symbols seen. This stack contains `BaselineStructure` objects rather than `Context` objects so that updating the whole structure is easier during the recognition process (see algorithm in Appendix B).

The implementation of the data set creation and of the symbol classification is explained in Appendix A. Appendix B provides the code for the main algorithms of the developed system.

Chapter 6

Results and Evaluation

We do not perform the complete recognition. Instead, we calculate confidence values for symols' class and for the structure recognition with missing information (the identity of symbols). It is difficult to present the results because we cannot really define a success rate. Evaluating such a system consists of three tasks. First, we should choose suitable parameters, and justify why they will illustrate the accuracy of the recognition process, and how they can reflect the performance of the proposed system. Then, we should have at our disposition a test set, which content has not been seen in the training phase. Building a test set involve collecting mathematical expressions and storing the expected recognition. Finally, we actually perform the tests. We use the expressions in the test set, and compare the actual recognition to the expected one. The results are expressed with the parameters defined before.

6.1 Parameters used for Evaluation

This project only aims to achieve part of the mathematical expression recognition. Indeed, we do not recognize the symbols but rather put them into classes, given the structure of the formula. On the other hand, we cannot expect a perfect recognition of the structure, provided that the symbols identities are not known. Therefore, we should not only count the mistakes of the system, but also estimate how confident it is on the expected recognition. We have defined five parameters to evaluate our system. Three of them give information about recognition errors, while the other two express the correctness of this recognition.

6.1.1 Recognition Errors

We implemented three main tasks for the recognition. They are the symbol classification, the parenting, and the relationship recognition. This can lead to three kinds of mistakes: the misclassification of a symbol, an incorrect parenting, or a wrong identification of a relationship. As a consequence, we defined three error parameters:

- Symbol misclassification, $e_s = \dfrac{\text{number of misclassified symbols}}{\text{number of symbols}}$
- Parenting error, $e_p = \dfrac{\text{number of symbols with wrong parent}}{\text{number of symbols}}$
- Wrong relationship identification, $e_r = \dfrac{\text{number of wrong relationships}}{\text{number of symbols}}$

Since the parenting and the identification of the relationships are done at the same time, it seems likely that a parenting error leads to a relationship error. However, the opposite is not necessarily true, as the relationship classifier might be wrong for a right pair of parent and child symbols.

6.1.2 Correctness Scores

Since we do not perform the whole recognition, the aim is not necessarily to achieve a correct recognition, but to have a high confidence in the expected recognition. During the design, we ensured that the final classification is not a crisp one. We want the system to be flexible and to leave ambiguity, so that the results can be used to efficiently recognize the characters involved. The output of the system is not only a class for each symbol and relationship, but also a set of confidence values for each of them.

It represents how confident the system is about the results it yields. The recognition error scores emphasis the cases where the system is wrong. The purpose of the correctness scores is to estimate how good or bad the system is. Let us present an example. On Figure 6.1, one can see (a) a mathematical expression, (b) the corresponding bounding boxes, input of the recognition process, (c) the distribution of confidence for the class of the symbol 'B'. This symbol is classified as 'small' although the actual class is 'ascending'. However, we can see that the confidence as 'ascending' is almost as high as the confidence as 'small'. Given that this symbol has little context, we can consider that the classification is still good.

For each symbol, we defined two correctness scores. The symbol correctness score, C_s, evaluates how confident the system is about the actual symbol class. The relationship correctness score, C_r, concerns the confidence about the actual relationship.

6.1.2.1 Symbol Correctness

The symbol class is inferred from the output of several classifiers, as explained in the previous chapters. The final distribution takes into account the shape of the bounding box, and the context of the symbol. Further uncertainty is introduced in the case of a lack of context. This could result in an error of the symbol classification, even though the confidence in the actual class is quite high, as seen on figure 6.1.

In this project, we consider four symbol classes. The sum of confidences over all classes being one, basic arithmetic gives us the following conclusions:

- No more than one class can have a confidence over 0.5
- No more than two classes can have a confidence over 0.34
- No more than three classes can have a confidence over 0.25

The symbol correctness score is not the confidence of the found class, but the confidence of the expected class. The higher this figure is, the more successful the system has been. In particular, due to the previous conclusions, and to the fact that uncertainty is willingly introduced, we can conclude that, if this score is:

- over 0.5, the classification has been excellent, no symbol error is possible.

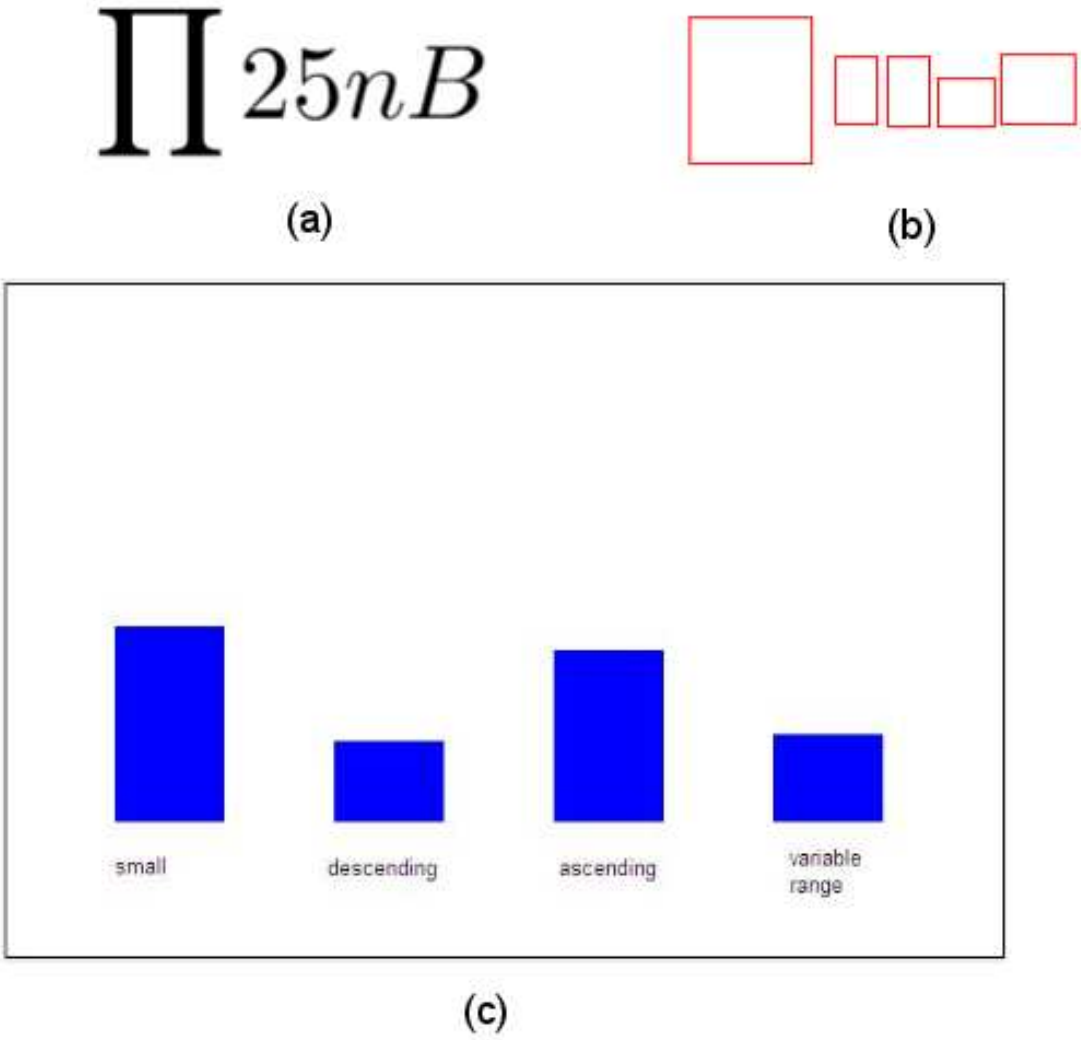

Figure 6.1: A soft symbol classification: (a) a mathematical expression, (b) bounding boxes, (c) confidence for the class of B

- over 0.4, the classification has been very good, because there is either no symbol error (very likely) or the gap between the confidence on the recognized class and the confidence on the actual class is very thin.
- over 0.34, the classification is good, because the actual class is at least the second most likely, if not the most likely.

6.1.2.2 Relationship Correctness

The same reasoning leads to the relationship correctness score. Even though there are five relationship classes, the same numbers (0.5, 0.34 and 0.25) hold to evaluate the results. However, we are more demanding on the relationship. Indeed, the purpose of the symbol classification is to help the actual symbol recognition task in a next step, whereas the relationship classification aims at performing the structure recognition. Therefore, we

expect to have better scores for the relationship correctness. However, we have to keep in mind that the structure recognition is more difficult with bounding boxes only, and possibly wrong symbol classes, than with the actual identity of the symbols.

Another issue arises when we calculate this score. The relationship recognition is tightly linked to the parenting of symbols. Thus, an error on relationship can be due to a parenting error. Then, it makes no sense considering the confidence of the actual relationship class with the wrong parent. Therefore, we differentiate two cases. If the parent is right, the relationship correctness is the confidence of the actual relationship class with this parent. If a parenting error is detected, the relationship correctness is obtained by calculating the confidence of the actual relationship with the actual parent.

6.1.3 Aggregating the Scores

These two kinds of scores represent two levels. The error scores can be calculated at a global level, that is considering a whole expression or set of expressions. The correctness scores are calculated at a symbol level. These scores should be aggregated, as we want to present results at different levels:

- the performance for each expression
- the performance for each test set
- the overall performance for all test sets

In the implementation, we defined a `Statistics` class of objects. It represents the performance for one expression. The information stored in this object is:

- the number of symbols
- the number of symbol class, relationship and parenting errors
- the global symbol and relationship correctness scores

as well as the misclassification by classes, a confusion matrix, and so on.

e_s, e_p and e_r are obtained by dividing the corresponding number of errors by the number of symbols. C_s and C_r are directly stored by taking the arithmetic mean of the corresponding symbols.

To calculate the performance at a higher level (test set or overall), we defined a method `merge( Statistics)` in this class. We compute the aggregated scores in the following way:

- Number of symbols $N = N^{(1)} + N^{(2)}$
- Number of errors $n_x^{err} = n_x^{err,(1)} + n_x^{err,(2)}$, for $x \in \{s, p, r\}$
- Correctness $C = \dfrac{N^{(1)} \times C^{(1)} + N^{(2)} \times C^{(2)}}{N}$

and similar methods for the other parameters.

6.2 Design of the Tests

To achieve a good evaluation, we defined two tools. First, we will present the test sets, built by hand. Then, we will show how we aim to compare our results to human classification.

6.2.1 The Test Sets

The test sets are built by hand. Most of them contain expressions which are in the scope of the project. That means, they respect some limitations on the complexity (0, 0.5 and 1), the form (LaTeX), and the symbols they contain. Some however exceed these limitations, and allow us to test the flexibility of the system.

Our GUI enable the opening of several expressions, their full labelling, and the ability to export the interpretations. The test sets are made of several files:

- for the LaTeXtest sets, we have a TeX file containing all expressions and we produce an XML interpretation file after the manual labelling, within the system.
- for non-LaTeXtest sets, we have a set of image files, and an XML interpretation file.

We also implemented the ability of the system to produce a text output to create a Latex table containing the original expression, the expected interpretation, the actual one, and the score, in order to be copied and pasted in the 'Results' appendix of this dissertation.

Each test set contains five to ten expressions. There are several motivations for it. It allows to split the whole test set into units with similar content (e.g. same complexity). Moreover, if a mistake is made in the manual labelling, correcting it requires to re-process ten expressions rather than a hundred.

Overall the test set contains 95 expressions, and 570 symbols. It is divided into 10 test sets:

- testset-0-1 and testset-0-2
- testset-1-1 and testset-1-2
- testset-2-1 and testset-2-2
- testset-3-1 and testset-3-2
- testset-NL
- testset-HW

The limitations on the number of symbol and relationship classes make it harder to write meaningful expressions. The formulae in the test set may not always represent expressions of the real world, but we claim they are good to test our algorithms.

Test sets 0-1 to 3-2.

They are LaTeXgenerated test sets containing 10 expressions each. The first number corresponds to the complexity.

- 0: order 0
- 1: order 0.5
- 2: order 1
- 3: order over 1

So the first three are within the scope, whereas the last one is outside. The second number is 1 for relatively simple expressions, and 2 for more difficult ones, also including some unknown (by the classifiers) symbols, such as Greek letters or different variable range symbols (e.g. $\bigwedge$, $\bigvee$ or $\bigoplus$).

Test set NL.

NL stands for Not Latex. It corresponds to ten expressions of different complexities generated by other equation editors (e.g. MS Word, Google Docs). It aims at testing the flexibility of the system regarding other forms of printed expressions.

Test set HW.

HW stands for handwritten. It contains five handwritten expressions. It is for testing the flexibility regarding a wider variation of the writing style. It is obviously the hardest task.

The content of these test sets can be found in the 'Results' appendix.

6.2.2 Comparison with Human Labelling

The Oxford dictionary defines artificial intelligence as "the theory and development of computer systems able to perform tasks normally requiring human intelligence" [1] . Thus, it seems natural to compare the performance of our system to the ability of humans to perform the same task in the same conditions.

To do so, we have to ask people to take the time to manually label expressions. We had little time to get enough results, so we specified how the inquiry should be presented.

1. It should get as many answers as possible in a very short time (one week)
2. People should see an interest in answering it.
3. It should be clear and simple, because people would not take the time to answer if they do not understand.
4. It should be quick to answer.

[1] Oxford Dictionary, Oxford University Press, http://oxforddictionaries.com/view/entry/m_en_gb0042040 , Aug. 2010

As a consequence, the appropriate solution seemed to be in the following form:

1. On the Internet, so it is easy to reach a lot of people in a short time.
2. It is a quiz, with a score at the end, so that it is somewhat fun to answer.
3. It begins with a short explanation of the project and of how to answer the quiz, both in English and in French, to reach more people.
4. It uses Web 2.0 technologies, to make it more intuitive to label expressions.

The implemented website, which can be found at `http://mscproject.tbluche.com/` , is made of four pages:

- A presentation of the project and of the quiz, with illustrating pictures (Figure 6.2)
- A presentation of some examples (Figure 6.3)
- The 'quiz' page
- The 'results' page, which purpose is to store the answers in a database, and display the score obtained.

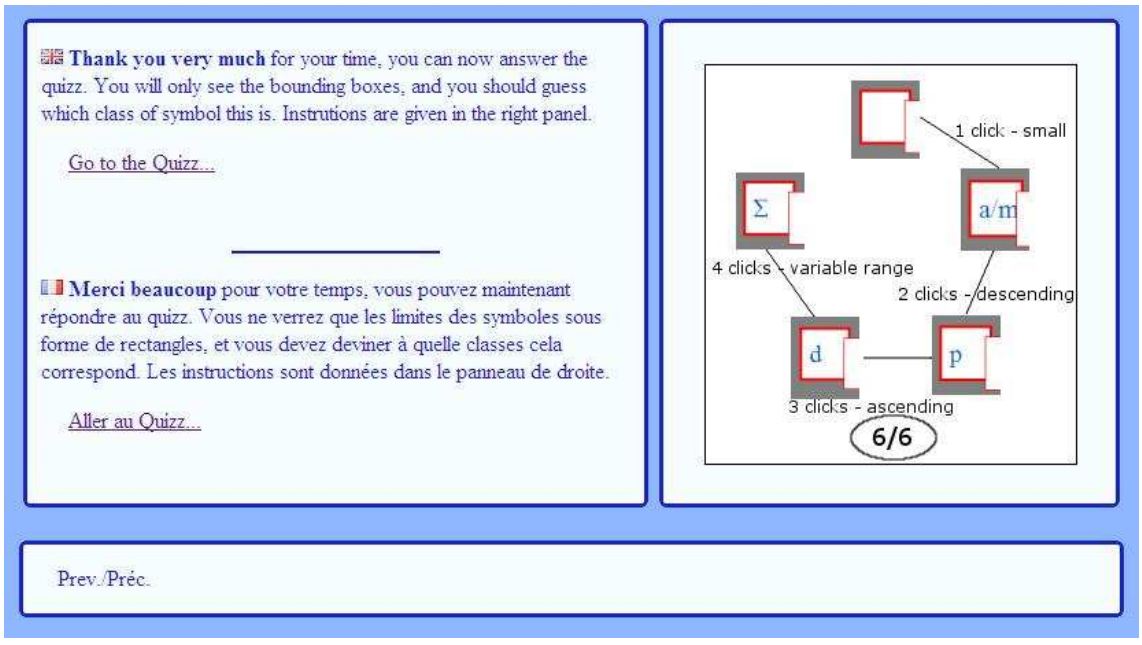

Figure 6.2: Presetation of the Project

The quiz should reflect the conditions in which the recognition is done by the system, that is with bounding boxes only. In order to keep the quiz quick and simple, we focus only on one task, that we consider the most difficult. So the user will not be asked to segment the image, to parent symbols, or the classify relationships, but only to label symbols (bounding boxes) with the correct class.

Building the Quiz Page

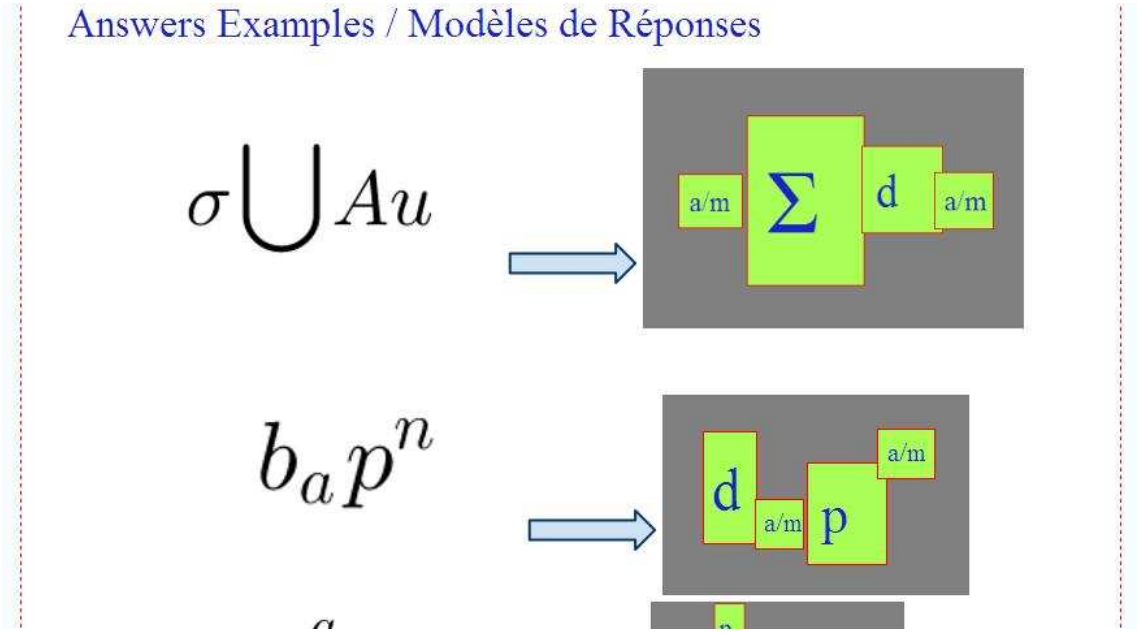

Figure 6.3: Presentation of Examples

The quiz page should be both easy for us to build, and easy for the user to understand and to answer. We used Cascading Style Sheets (CSS) and HTML `div` tags to display expressions (symbols bounding boxes). It is directly implemented in the system. As explained before, the GUI can handle several expressions. We added a menu that allow to create the HTML page. Using the JDOM framework, it creates a `div` for each expression, adjusted to the expression dimension, containing a `div` for each symbol, with an absolute positioning and scaling. A screenshot of the produced web page is shown on Figure 6.4.

Labelling Mechanism

To label the expression, the user has to select the class of each symbol in a list. However, choosing the class by its name in a list is not always as straightforward as it seems. Rather, we specified representative symbols for each class. It is a and m for the class 'small', p for 'descending', d for 'ascending' and $\sum$ for 'variable range'.

We used the jQuery framework to implement the mechanism. When the user clicks on a box (i.e. a symbol), 'a/m', 'p', 'd' and '$\sum$' are successively displayed in the box. It makes the labelling process easy, especially as it produces a visual result allowing the user to check whether their labelling corresponds to what they meant. An example of an expression being labelled is shown on Figure 6.5.

Getting the Answers

The list of classes is compared to the expected list, to compute the score, and is stored in a database.

To get as many answers as possible, I sent the link to the quiz to the students of the MSc in Computer Science of the Computing Laboratory and the students of my previous school,

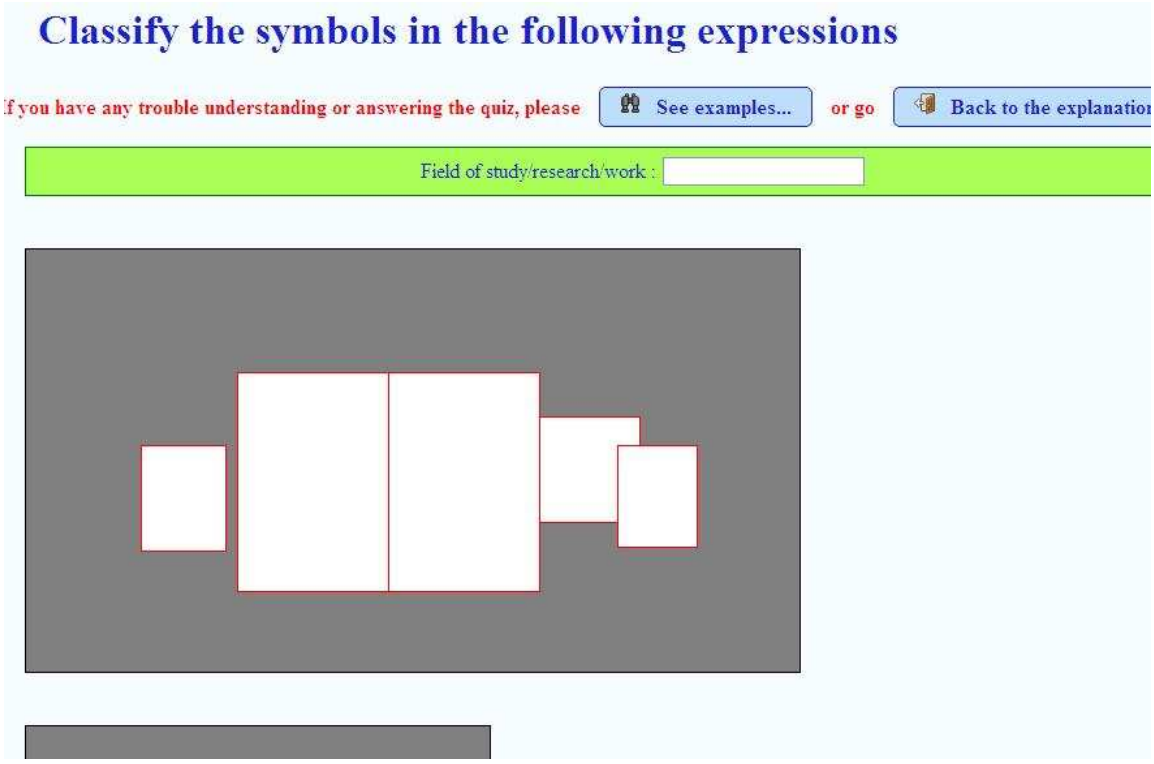

Figure 6.4: The Quiz Page

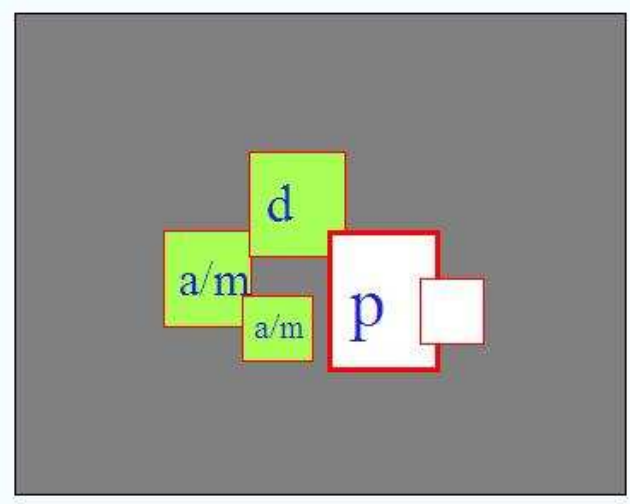

Figure 6.5: An item in the quiz being completed

Supélec. I also posted it on my profile on several virtual networks, such as Facebook.

The quiz contains 10 expressions, and 69 symbols. They are taken from the test set, and they can be found in the appendix about results.

In one week, 379 people visited the website, and 101 answered the quiz. Amongst them, only 50 labelled all symbols. Some however, labelled only several expressions. Each expression was labelled by at least 55 people.

Table 6.1: Performance for Different Iterations

Number of iterations	1	2	5	7	10
Symbol classification	76.49%	77.02%	77.19%	77.37%	77.19%
Relationship classification	86.32%	90.53%	90.88%	90.88%	90.70%

Table 6.2: Symbol Recognition Rate

Class	Number of Symbols	Correctly Classified	Score
Small	225	212	94.22%
Descending	68	57	83.82%
Ascending	218	137	62.84%
Variable range	59	34	57.63%

6.3 Performance of the System

A comprehensive presentation of the performance of our system on the test sets can be found in the 'Results' appendix. In this part, we summarize the results for each test sets. We tried to vary the number of iterations to compare the corresponding results. We will present some examples and show how the system performed on them. Besides, we will try to emphasis the extent to which our original hypothesis is verified. Indeed, our goal was to show that the mere structure recognition could help classify the symbols, and that symbol classes were sufficient to yield good results in the structure recognition. First we will present the results. Then, we will evaluate them.

6.3.1 Presentation of the Results

We tested the system with 1, 2, 5, 7 and 10 iterations, and the results are presented in Table 6.1.

The difference is not very high. In order to keep the recognition as quick as possible, we chose 5 iterations. Given that the average number of symbols per expression in the test set is about 6, we decided to choose a smaller number of iterations. In the following, the results are presented, for each task, after 5 iterations.

6.3.1.1 Overall

440 of the 570 symbols (77.19%) have been successfully classified. The average symbol correctness is 0.382. It means that on average, a symbol is often well classified, or the actual class is considered as the second most likely. The recognition rate per symbol class is shown in Table 6.2.

There have been 32 parenting errors (success rate of 94.39%), and 52 relationship errors (9.12% of misrecognition). The overall relationship correctness is however very high (0.797). A summary of the recognition is shown on Figure 6.6.

Here are some examples within the scope of the project, of different complexities and well recognized:

$$1a2b3p \quad tan\pi \quad b_a p^n$$
$$y_l^n z q_p \quad e^{2ln2} \quad e^{\prod lnx}$$

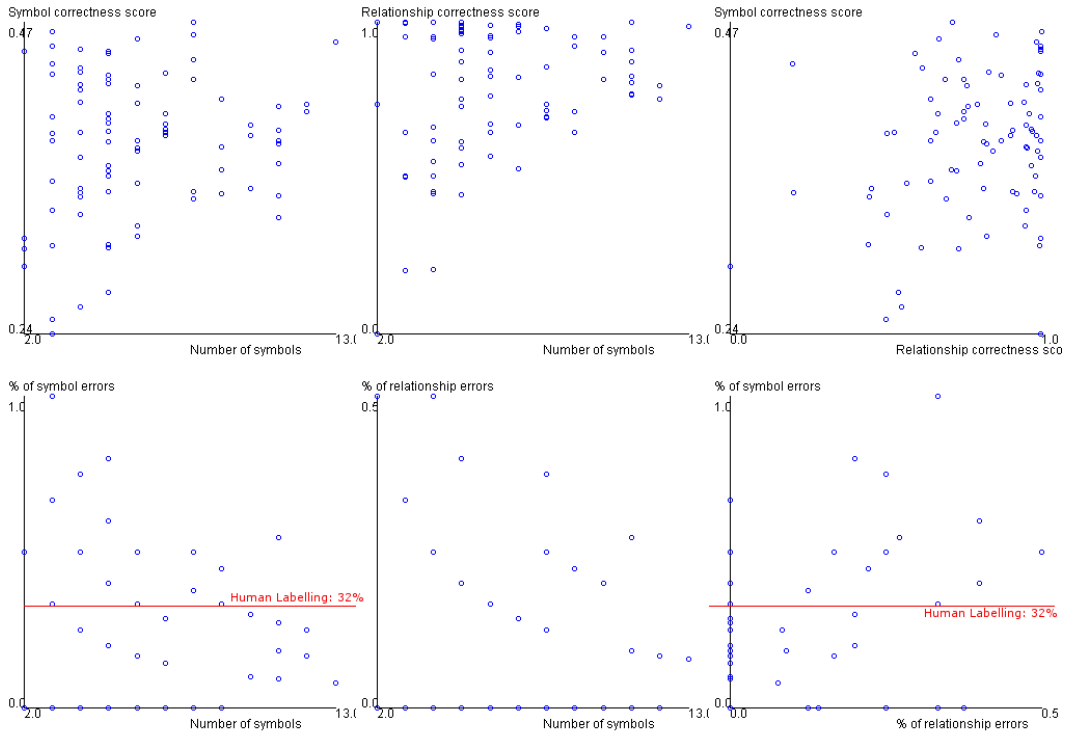

Figure 6.6: Overall Recognition

Some examples are outside the scope but yield good results:

- Right complexity (order 1) but with symbols unknown by the training set

$$\varepsilon_{\alpha p}^{\Gamma \pi}$$

- Higher complexity

$$d^{a_{23}^{b}} v b_{n}$$

- Non Latex form or handwritten (almost perfect symbol classification and perfect structure recognition), see Figure 6.7.

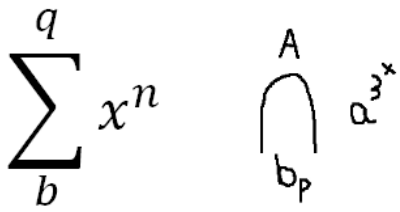

Figure 6.7: Example of Expression Well Recognised

6.3.1.2 Per Test Set

The test sets have different contents. The complexity, content, and form of the expressions are different, and some examples are not in the scope of the project, but used to

Table 6.3: Test Sets Scores

Testset	Symbol Classification (%)	Parenting (%)	Relationship Recognition (%)	Symbol Correctness	Relationship Correctness
testset0	66.67	95.24	84.52	0.362	0.537
testset1	87.78	95.04	95.04	0.404	0.922
testset2	81.63	95.92	90.48	0.392	0.777
testset3	78.79	90.91	91.67	0.390	0.922
testsetNL	61.02	94.92	89.84	0.347	0.826
testsetHW	70.37	96.30	92.59	0.381	0.852

test the robustness of our system. Therefore, presenting the results for each test set is useful. Table 6.3 summarizes the performance for each test set, including the success rate of the symbol classification, the parenting and the relationship identification, as well as the correctness scores.

The different distributions of errors and correctness for each test set are presented on Figure 6.8.

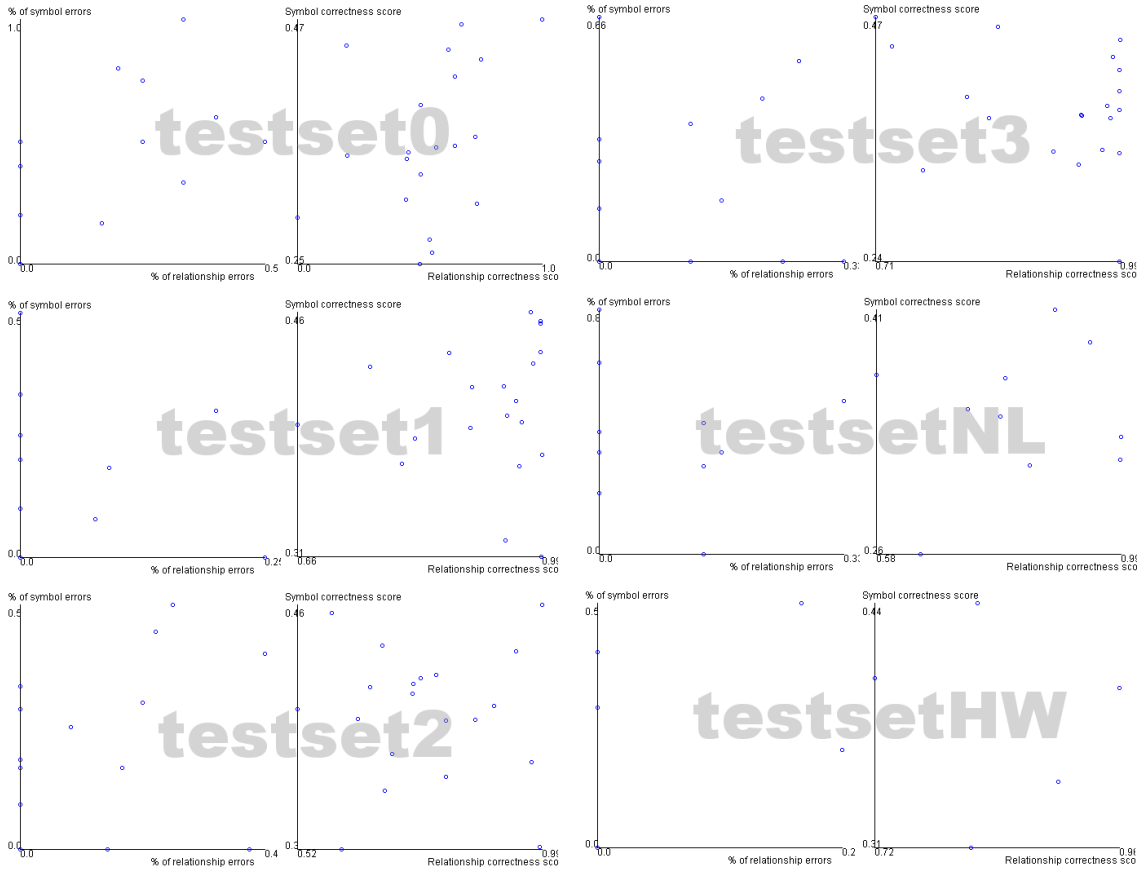

Figure 6.8: Recognition by Test Set

Table 6.4: Average Scores of Human Labelling

Equation	Number of symbols	Test set	Number of answers	Average score (%)
1	5	0-2	97	59.82
2	10	1-1	78	65.63
3	7	1-2	74	68.92
4	5	2-1	64	55.80
5	9	2-2	59	74.92
6	6	2-2	59	63.28
7	8	3-1	59	63.56
8	7	3-1	57	61.18
9	7	NL	56	59.16
10	5	HW	57	75.69

Table 6.5: Human Confusion Matrix

classified as ¿	small	descending	ascending	varrange
small	1200	158	101	20
descending	213	300	157	44
ascending	411	287	815	150
varrange	49	20	23	617

6.3.1.3 Human labelling

The human labelling only concerns the symbol recognition. The data set for human labelling contains 10 expressions (see 'Results' appendix). Amongst the 50 persons who completely answer the quiz, the average score was 67.94%. Table 6.4 presents the scores for each expression, taking into account only complete answers, that is when all the symbols of the expression have been labelled.

We can see in Table 6.5 how people labelled the symbols. We notice that descending and ascending symbols are more often misclassified than small symbols and variable range.

6.3.2 Evaluation

In this part, we will analyse the results, and try to explain the mistakes of the recognition. We will show that the results are good, and that the hypothesis is verified. The test sets contain both expressions within and outside the scope of the project. Thus, we can first explain both how good and how flexible is the system.

6.3.2.1 Scope of the Project and Flexibility

Our project focuses on the importance of the context in the symbol classification, and has some limitations on the form and complexity of the expressions. As we foresaw, symbols in formulae of order 0 are not easy to recognize, because few context is available (maximum 2 symbols, a parent and an inline child). We recorded 4 parenting errors amongst the 84 symbols and the 20 expressions. In these parenting errors, we recorded 3 mistakes on the parent's class and 1 on the child's class.

We reached the best results for expressions of order 0.5 (testset1), which are the expressions in the scope of the project. There have been 16 symbol misclassifications and 6

parenting errors for 121 symbols. However, the correctness scores, which show how confident the system is about the right interpretation are high (0.404 for symbols and 0.922 for relationships).

The results are lower for testset2, which contain expressions of order 1. They are actually expressions of order 0.5 with nested expressions of order 0. The misclassifications due to the lack of context in the nested expressions might explain this lower performance. Testset3 contains expressions of order higher than 1, outside the scope of the project, but the results are still good, proving the flexibility of the system regarding the complexity.

Results for testsetNL are not very good. In particular, it looks like the ascending and descending symbols do not have the same shape (and baselines) with MS Word and Google Docs typesets as with Latex. Indeed, 61% of the symbol misclassifications concern ascending symbols and 17% concern descending symbols. The flexibility is particularly observable in testsetHW, containing handwritten formulae. Even if nearly 30% of the symbols are misclassified, the symbol correctness is promising (0.381 on average, 0.31 minimum, 0.44 maximum). Finally, the system is flexible regarding unknown symbols. We reach some good classification of Greek letters for example, which are not present in the training sets.

6.3.2.2 Evaluation of the Structure Recognition

We want a fast but robust structure recognition. Therefore, we use a one-pass algorithm with little backtracking. There are two challenges in our approach. First, it seems easier to test every pair of symbols and keep the most likely, whereas we stop investigating as soon as a confidence for a relationship and a parent is higher than the threshold. Moreover, the misclassification of symbols can jeopardize the recognition of the structure.

Indeed, we record overall 32 parenting errors. We found that in the 32 pairs of symbols which should be linked, there are 15 misclassifications of parents, and 6 misclassifications of children. Amongst the 52 mistakes on relationships, 12 correspond to an error on parenting, 26 involve a misclassification of the actual parent, and 24 a misclassification of the child. Provided that the relationship correctness is very high, nearly 0.8, we can conclude that the structure recognition is good when the symbols are correctly classified.

6.3.2.3 Analysis of the Symbol Classification

The second challenge of this project was to classify the symbols using only the context and the bounding box. The results of the recognition proved the importance of having a lot of context. On the other hand, with fewer context, some similarities between symbol classes can be a source of mistake.

The Importance of Context

The symbol classification is successful in 77.19% of the examples in the test sets, and this score reaches more than 87% for testset1. The recognition is even more efficient than human labelling in the same conditions, except for handwritten expressions.

Table 6.6: Symbol Confusion Matrix

classified as ¿	small	descending	ascending	Variable range
small	212	4	3	6
descending	4	57	2	5
ascending	50	26	137	5
variable range	5	13	7	34

130 of the 570 symbols are misclassified. The mistakes are mainly made in the ascending class (63% of all mistakes). The errors for each class are:

- Small: 13 (10%)
- Descending: 11 (8%)
- Ascending: 81 (63%)
- Variable Range: 25 (19%)

More importantly, we observe that 93.8% of the mistakes are made for symbols having only one or two pieces of context. The distribution of errors over the different amounts of context are:

- 1 piece (a parent and no child (last symbol of a baseline), or one child and no parent (first symbol of the expression)): 59 (46%)
- 2 pieces of context (one parent and only one child): 63 (48%)
- 3 pieces of context: 4 (3%)
- 4+ pieces of context: 4 (3%)

Sources of Mistakes

To try to understand the possible sources of mistake for the system, we build a confusion matrix for the symbol recognition, shown in Table 6.6.

In the data analysis, we noticed that variable range and descending symbols have a similar ratio. Moreover, their baselines positions are relatively close. When there is no or little context to disambiguate it, a variable range symbol is often misclassified as descending, as in

$$\varphi \bigvee \bigwedge T\, p$$

where $\bigvee$ and $\bigwedge$ are classified as descending.

We also notice that capital letters have a similar ratio as small letters, although being ascending symbols. Added to the fact that the baselines are the same, it can explain the large number of misclassifications of ascending symbols as small ones.

Finally, we saw in the data analysis a serious overlap in the ratio distribution of ascending and descending. It means that some ascending letters, such as h have a typical ratio of the descending class, and are often misclassified if there is not enough context. For example h is well classified in

$$\bigcup_{H} h_m^n \bigcap_{t}^{T} x_b$$

but not in

$$m_{g^h}$$

The quiz proved that humans have the same difficulties on these symbols in the same conditions as our system.

Chapter 7

Conclusions

To conclude this dissertation, we will first explain how our approach allowed us to get interesting results. Then, we will evaluate to which extent the goals are met. We will point out the cases where the proposed system works well, but also the limitations. This will lead us to finally expose our ideas for further development of the proposed system.

7.1 Findings

We formulated the hypothesis that the structure recognition could be performed without the symbol identities. Besides, we assumed that the bounding box of symbols, and the context available could help recognize the symbol class.

Although a lot of exiting techniques for mathematical expression recognition define rules (such as grammars), we wanted to have a flexible system. Mathematical notation can be different from one research field to another, and evolves a lot. Our system should adapt itself to those trends, but also to the variation in the writing style. To achieve this goal, we developed a system based on machine learning techniques. Each component has been chosen to suit the best its function. We created a training set, and a test set, to train and build a multi-classifier system.

An iterative algorithm has been developed to exploit the mutual constraints between the structure and the type of symbols involved. Although the symbol classification consists in classifying each symbol individually, the structure recognition is more complex. The relationships between symbols must be found and identified. We implemented a one-pass algorithm including very few backtracking, that yielded a fast recognition of the structure.

The recognition returns confidence values for each symbol and relationship, rather than a crisp interpretation. Presenting the results with confidence values should allow an easy utilization of the system as part of a bigger one, performing the whole recognition. These confidence values are also used to define scores, which give an idea of how good the proposed system is.

We built a graphical user interface to make both the design of the system and the usage of the expression recognizer quick and easy-to-use. It was also particularly helpful for the writing of this dissertation. Finally, we designed a quiz for human users to label expressions on the Internet. It allowed us to compare the performance of this intelligent system to human expertise.

We defined a scope for this project. The results in that scope were very good. In particular, we showed that symbols can be successfully classified when enough context is available. In the case of a symbol misclassification, we noticed that the confidence value for the actual class was quite high too.
The structure recognition was fast (about 10ms) and its results were good. Indeed, a printed expression is for humans to globally see and understand. Our approach showed that machine learning techniques make the structure recognition possible by comparing symbols two by two.

During the test sets design, we included expressions exceeding the scope of the project, to test the flexibility of the system. Given the format of our training set, we were surprised to obtain such good results.

7.2 Evaluation

The aims of this project are met. We proved that context could indeed help the classification of symbols, even if only their bounding boxes are used. The fuzzy regions and baselines helped well the construction of a fast and efficient structure recognition. Moreover, we saw that a misclassification of a symbol do not necessarily compromise the structure recognition.

Furthermore, we had to design our own training set. We kept it very simple, and simulated variations in the size and positions of symbols. This simplicity did not reflect the reality of mathematical expressions, but the association of the classifiers trained from the data set yielded good results, even on complicated formulae, proving the robustness of the proposed system.

The results of this approach are good, provided that only the bounding boxes of the symbols are used. However, the analysis of the results showed that symbols do not always have enough context to be successfully classified. This can later lead to mistakes in the structure recognition.

We have to keep in mind that the interesting part of the results is the confidence value, because our system is only a first step towards a global mathematical formula recognition. However, some ideas to get around the limited performance due to the lack of context are exposed in the next section.

7.3 Ideas for Further Development

The results have proved that our hypothesis was worth investigating. The performance concerning the symbol recognition given the context were correct, and the structure recognition method was promising, even for some complicated, or handwritten structures. However, the serious mistakes done by the proposed system in some situations show a need for improvement. In particular, the data set we used was probably too simple to be adapted to all situations. The structure recognition is fast, but can lead to errors which jeopardize the recognition of the remaining relationships. Some improvement is also necessary for this part. Finally, the system should be completed to include all symbol classes, relationships, and different writing styles, as well as the actual symbol recognition.

7.3.1 Improving the Training Set

The training sets are simple enough to be built automatically. However, they do not represent the possible complexity of mathematical expressions. A training set built with a higher diversity may give a more complete image of the different structures. Some mistakes, both in the structure and the symbol recognition could be avoided.

Secondly, during the recognition, the mistakes done in one task can give way to mistakes in the other task. The system handles uncertainty, but the components have not been trained with uncertain examples. Finding a way to represent uncertainty in the training sets, or including some mistakes could add to the flexibility and robustness of the system. The '$Infty$' project data set could be processed and used to train our system if we define and add more symbol classes and relationships.

7.3.2 Improving the Recognition

The recognition would of course benefit from a better data set. In this project, the data was exploited locally, by fast algorithms. A global vision of expressions may help their recognition, as a human user naturally disambiguates the interpretation by seeing the expression globally.

The context of a symbol is only made of the direct children and the parent, and the bounding boxes convey very little information. The relationships are links between two symbols, whereas the structure recognition often involve grouping symbols. For examples, some techniques extract the symbols in the dominant baseline by looking to all symbols' positions. We consider pairs of symbols, out of a more global context, because it is adapted to the possible variations in the writing style. This is especially true for handwritten expressions, in which baselines seldom have fixed positions.

For the symbol recognition, the context is often small, and this leads to misclassifications. We could try to build a system which considers a wider context. A global part might keep track of expression-level information, such as the different baselines, the size of the symbols in these baselines and so on.

For the structure recognition, it might be useful to consider more than pairs of symbols. The position of baselines is partially exploited in our implementation, but could be improved by considering baselines more globally. Having a simple and fast algorithm was our goal, but it may still be achieved if we consider a wider context in the structure recognition as well. For example, we can try to look at the context of the symbols, when a pair is processed.

Finally, the uncertainty about symbols' classes is used in the structure recognition, through the fuzzy baselines and fuzzy regions. The uncertainty about parenting links and relationships is not exploited in the symbol recognition in our project. Keeping uncertainty in the structure often means making the expression tree more complex. It should indeed store the different possible interpretations. A search algorithm is then needed to find the best one. However, investigating how to keep this uncertainty, at least for the symbol classification may result in a better performance.

7.3.3 Extending the Proposed System

Besides the creation of a more comprehensive data set, the system needs some improvement to be a marketable system. First, we have to consider all symbol classes and all kinds of relationships. One can also try to have a system to learn new relationships and classes, for the system to be even more adaptable. We have to think of a way to handle lines, such as fraction lines, overbars, minus symbols, and so on. The lines have a short height, hence a big ratio. Their form may lead to problems, especially when creating the fuzzy regions. The small height will surely produce small regions.

We did not focus on the segmentation. A complete system should include a segmentation algorithm able to tell that '=', for example, is one symbol. If we are to use the bounding boxes and context only, this task can be very hard, but important.

The system aims to be flexible, so we used some machine learning techniques. Some parts however are not learnt, such as the position of baselines. It may be worth investigating the possibility to have a system completely driven by machine learning artefacts. A more general data set could make it possible. It might also be useful to return a set of interpretations, rather than one interpretation, with a set of associated confidence values. In order not to make the structure recognition too long and complex, using the interpretations at each iteration to build this set might be interesting. The convergence of the interpretations to the observed result seems logical. Therefore, having just a set of intermediate interpretations might not be informative. We may avoid the convergence by using random parameters in the recognition.

Finally, a complete system should include the recognition of the symbols. They have already been classified, and we believe that the recognition is made easier. Indeed, we need one classifier for each class, but the number of symbols to discriminate is smaller. The confidence values can be used to evaluate which is the most likely symbol, whereas a crisp classification would have been too constraining. Moreover, the identification of symbols can use the structure recognition results. For example, the fact that a symbol has some children may be used as features in the recognition.

Bibliography

[1] W. Aly, S. Uchida, A. Fujiyoshi, and M. Suzuki. Statistical classification of spatial relationships among mathematical symbols. In *Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on*, pages 1350 –1354, jul. 2009.

[2] Walaa Aly, Seiichi Uchida, and Masakazu Suzuki. Identifying subscripts and superscripts in mathematical documents. *Mathematics in Computer Science*, 2:195–209, 2008. 10.1007/s11786-008-0051-9.

[3] A.-M. Awal, H. Mouchere, and C. Viard-Gaudin. Towards handwritten mathematical expression recognition. In *10th International Conference on Document Analysis and Recognition, 2009. ICDAR '09.*, pages 1046 –1050, jul. 2009.

[4] D. Blostein and A. Grbavec. Recognition of mathematical notation. In H. Bunke and P. Wan, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 557–582. World Scientific, 1997.

[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] Shi-Kuo Chang. A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences*, 2(3):253 – 272, 1970.

[7] B. B. Chaudhuri and Utpal Garain. An approach for recognition and interpretation of mathematical expressions in printed document. *Pattern Analysis and Applications*, 3(2):120–131, 2000.

[8] Y. Chen, T. Shimizu, and M. Okada. Fundamental study on structural understanding of mathematical expressions. In *IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics, 1999.*, volume 2, pages 910–914, 1999.

[9] Y. Eto and M. Suzuki. Mathematical formula recognition using virtual link network. In *6th International Conference on Document Analysis and Recognition, 2001. Proceedings.*, pages 762 –767, 2001.

[10] C. Faure and Z. X. Wang. Automatic perception of the structure of handwritten mathematical expressions. In *Computer Processing of Handwritting*, pages 337–361. World Scientific, 1990.

[11] Pascal Garcia and Bertrand Coüasnon. Using a generic document recognition method for mathematical formulae recognition. In Dorothea Blostein and Young-Bin Kwon, editors, *Graphics Recognition Algorithms and Applications*, volume 2390 of *Lecture Notes in Computer Science*, pages 236–244. Springer Berlin / Heidelberg, 2002.

[12] R. Genoe, J.A. Fitzgerald, and T. Kechadi. An online fuzzy approach to the structural analysis of handwritten mathematical expressions. In *2006 IEEE International Conference on Fuzzy Systems*, pages 244 –250, jul. 2006.

[13] Ming-Hu Ha, Xue-Dong Tian, and Na Li. Structural analysis of printed mathematical expressions based on combined strategy. In *2006 International Conference on Machine Learning and Cybernetics*, pages 3354 –3358, aug. 2006.

[14] John C. Handley, Anoop M. Namboodiri, and Richard Zanibbi. Document understanding system using stochastic context-free grammars. In *ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 511–515, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Stephane Lavirotte. Optical formula recognition. In *ICDAR '97: Proceedings of the 4th International Conference on Document Analysis and Recognition*, pages 357–361, Washington, DC, USA, 1997. IEEE Computer Society.

[16] Hsi-Jian Lee and Min-Chou Lee. Understanding mathematical expressions using procedure-oriented transformation. *Pattern Recognition*, 27(3):447 – 457, 1994.

[17] Erik G. Miller and Paul A. Viola. Ambiguity and constraint in mathematical expression recognition. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 784–791, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[18] Taik Heon Rhee and Jin Hyung Kim. Efficient search strategy in structural analysis for handwritten mathematical expression recognition. *Pattern Recognition*, 42(12):3192 – 3201, 2009. New Frontiers in Handwriting Recognition.

[19] Masakazu Suzuki, Seiichi Uchida, and Akihiro Nomura. A ground-truthed mathematical character and symbol image database. In *ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 675–679, Washington, DC, USA, 2005. IEEE Computer Society.

[20] T. Suzuki, S. Aoshima, K. Mori, and Y. Suenaga. A new system for the real-time recognition of handwritten mathematical formulas. In *15th International Conference on Pattern Recognition, 2000.*, volume 4, pages 515 –518 vol.4, 2000.

[21] Ernesto Tapia and Raúl Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In Josep Lladós and Young-Bin Kwon, editors, *Graphics Recognition*, volume 3088 of *Lecture Notes in Computer Science*, pages 329–340. Springer Berlin / Heidelberg, 2004.

[22] 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.

[23] Xue-dong Tian, Si Wang, and Xiao-yu Liu. Structural analysis of printed mathematical expression. In *2007 International Conference on Computational Intelligence and Security.*, pages 1030 –1034, dec. 2007.

[24] K. Toyozumi, T. Suzuki, K. Mori, and Y. Suenaga. A system for real-time recognition of handwritten mathematical formulas. In *Proceedings. Sixth International Conference on Document Analysis and Recognition, 2001.*, pages 1059 –1063, 2001.

[25] Zi-Xiong Wang and C. Faure. Structural analysis of handwritten mathematical expressions. In *9th International Conference on Pattern Recognition, 1988.*, volume 1, pages 32 –34, nov. 1988.

[26] H.-J. Winkler, H. Fahrner, and M. Lang. A soft-decision approach for structural analysis of handwritten mathematical expressions. In *1995 International Conference on Acoustics, Speech, and Signal Processing, 1995. ICASSP-95.*, volume 4, pages 2459–2462 vol.4, may. 1995.

[27] Qi Xiangwei, Pan Weimin, Yusup, and Wang Yang. The study of structure analysis strategy in handwritten recognition of general mathematical expression. In *IFITA '09. International Forum on Information Technology and Applications, 2009.*, volume 2, pages 101 –107, may 2009.

[28] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1455–1467, 2002.

[29] Ling Zhang, Dorothea Blostein, and Richard Zanibbi. Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In *ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 972–976, Washington, DC, USA, 2005. IEEE Computer Society.

Appendix A

Implementation

A.1 Main Functionalities of the System - Use Case View

The most straightforward goal of the system is the recognition of an expression. However, we have identified in this dissertation several other useful functionalities. The most important ones are summarized in the diagrams on Figures A.1 and A.2. We do not enter in the details, since they have already been explained.

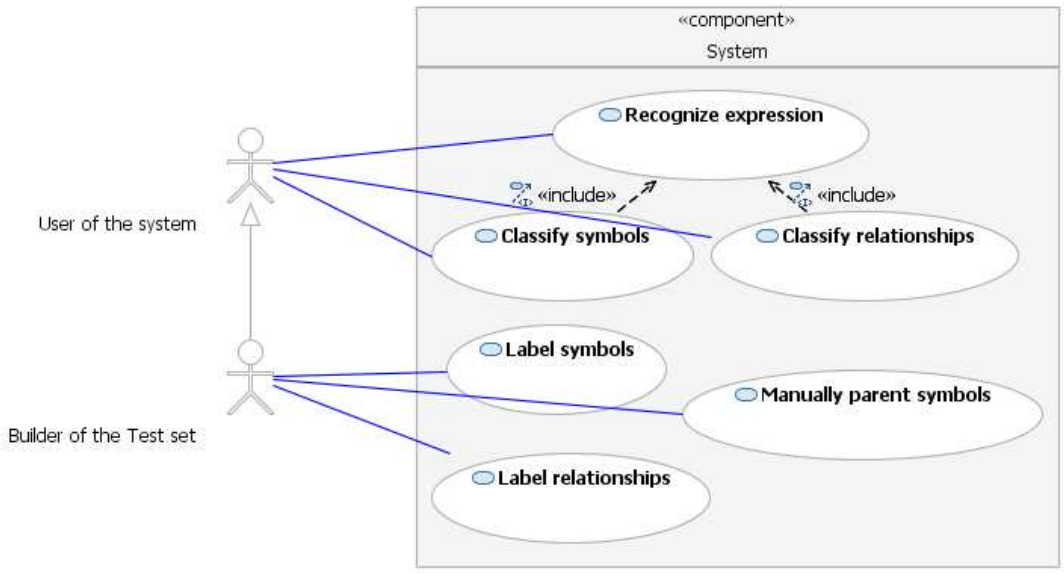

Figure A.1: "Classification" use cases

A.2 Package Organization

Figure A.3 represents the organization of the developed part of the application in different packages. They allow to separate objects according to their function, in a logical way. In the following, we explain the meaning associated with each package.

'Core' Package

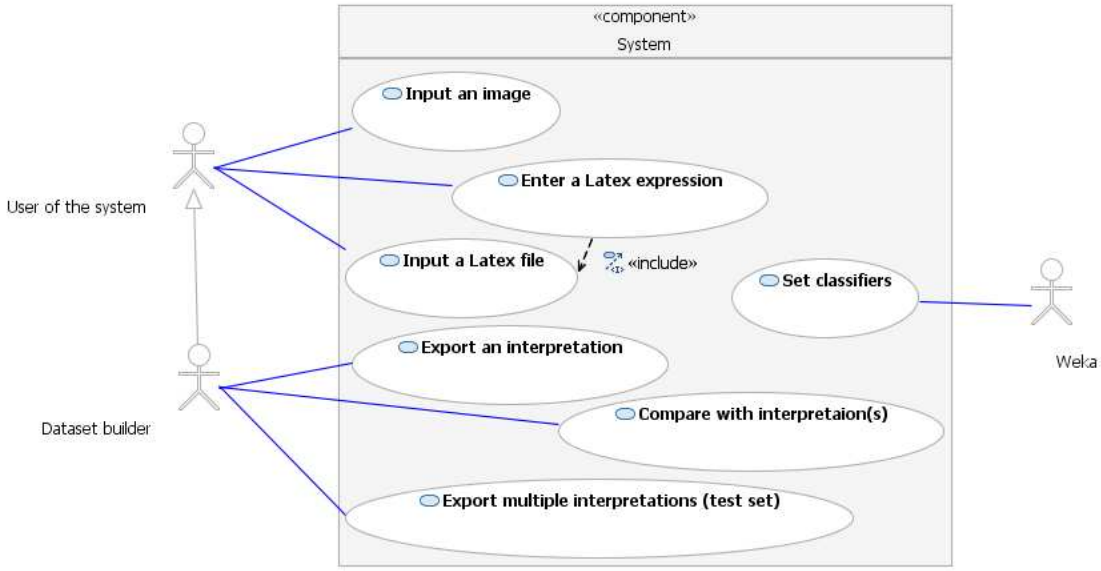

Figure A.2: "Import/Export" use cases

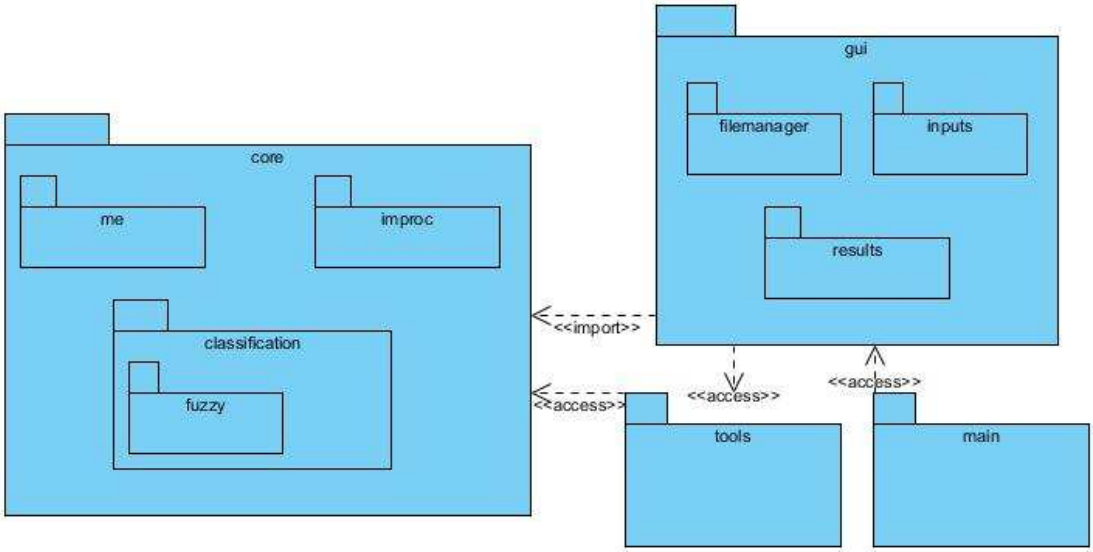

Figure A.3: Package Diagram

This package corresponds to the core of the system, that is the representation of an expression, and its recognition. It contains two sub-packages, which are:

- 'me' for the representation of the expression, and the symbols it is made of. It contains the methods to access all properties of the expression.

- 'classification' contains all artefacts allowing the classification of an expression. It includes the trained classifiers, but also the mechanism which recognizes the structure, and the representation of fuzzy regions.

'Tools' Package

This package contains tools corresponding to other useful functionalities. This includes the possibility to read or write XML documents, to represent and segment images, and to parse Latex files, and create binary images from a Latex expression.

'GUI' Package

The 'GUI' package focuses on every functionality related to the display of the data (e.g. expression, properties, classification, performance), and the actions available to the user or the designer. It contains basically all the objects corresponding to windows, panels, menus and buttons of the interface. It is made of several sub-packages:

- 'results' contains the windows regarding the display of the results and statistics
- 'inputs' contains the graphical tools which allow the user to input expressions (e.g. in the form of a LaTeXcommand, or handwritten)
- 'filemanager' contains the classes handling the workspace functionalities.

'Main' Package

The 'main' package is the entry point of the system. It includes a static class which contains the parameters of the system, such as thresholds or the number of symbol classes. The aim is to be able to modify easily some parameters, without having to change the code of the core classes. It also include the main function, which initializes the parts of the system and launches the interface.

A.3 Methods in the Expression Implementation

The content of package '*core::me*' has been presented in the 'Implementation Overview' chapter. In this section, we will explain the purpose of the most useful methods presented on the class diagram on Figure 5.2, on page 78. For further details, report to the API provided on the attached disk.

Class Symbol. For the calculation of the membership value in fuzzy regions, the symbol is reduced to a point. According to the definitions of the parameters, this point is the center of the left bound of the bounding box. It is retrieved by the method `getCentroid()`. `getCenter()` returns the position of the vertical center. The other methods essentially calculate parameters such as the height of the bounding box. `setID(int)` is used to modify the symbol id, in order to compare the expression to a stored interpretation.

Class Context. Some methods, such as `addChild(Context, int)`, allow the creation of the tree by the association of nodes. The methods implement some constraints such as the fact that a symbol must be one of the child of the node referenced as its parent. When relationships are undone, `clearForRelationship(int)` ensures that these constraints are maintained. Moreover, this object can access a symbol and its context. Therefore, it has to implement the methods for calculating the parameters (H, D, V, and so on), and for creating the input of the classifiers. To each symbol classifier corresponds a method like `buildClassifierInstance()`.

Class Expression. During the creation, symbols can be added directly with their bounding box (`addSymbol(int[])`). A `Context` object is created, with the corresponding `Symbol`. `process()` implements the whole iterative recognition algorithm. `compare(Expression)` allows to compare two expression trees, provided that they represent the same expression, that is with the same number of symbols, which have the same Ids. It is used to compare the results of the system to the test set. `assignIds(Vector<Integer>)` assigns the ids to the symbols for the expression to be comparable to another expression with the same ids.

A.4 Handling XML Files

The class `XMLCreator` allows to create or to read several kinds of files, which represent different purposes. It uses the JDOM framework. The ways it can be used are presented, for some, in the 'Algorithms' appendix, and for the rest, in the API and the code on the provided disk. On Figure A.4, one can see several XML files, generated (and, for some, read) by this class. They represent:

1. the interpretations of expressions
2. a list of files to open, corresponding for instance to a test set
3. a list of symbols' id to load for testing the system
4. the automatically generated quiz for the website

A.5 The Graphical User Interface (GUI)

In this section, we present more details about how the GUI is implemented.

A.5.1 The Menus

There are two kinds of menus. For the recurrent actions we display a set of buttons. A comprehensive list of all possible actions is found in the top menu.

Top Menu. The top menu is implemented by the `TopMenu` class. There are seven menus for different kind of actions:

- 'File' allows to open an input image, draw one, enter a Latex expression or close the program.
- 'Display' allows to show/hide all parenting links, all baselines, all fuzzy regions, the expression (to leave the bounding boxes only). From this menu, we can also re-open the results view or the file manager if they have been closed.
- 'Save' contains all actions involving exporting (the image, the image panel, the interpretation(s) or symbols list in XML)
- 'Load' allows to load different kinds of XML files (e.g. to compare an interpretation with the real one)
- 'Go to...' allows to navigate from an image panel to another
- 'Set class...' and 'Set relationship...' allow to label an expression.

1.
```
<expressions count="10">
    <expression width="126" height="129">
        <symbol id="6" class="1" rel="-1">
            <boundingBox>
                <xmin>43</xmin>
                <xmax>65</xmax>
                <ymin>63</ymin>
                <ymax>85</ymax>
            </boundingBox>
            <symbolClass>
                <class id="1">0.4303918844197614</cl
                <class id="2">0.1428140109883948</cl
                <class id="3">0.26488460479316966</c
                <class id="4">0.16190949979867414</c
            </symbolClass>
            <relationshipClass>
                <class id="0">0.0</class>
                <class id="1">0.0</class>
                <class id="2">0.0</class>
                <class id="3">0.0</class>
                <class id="4">0.0</class>
            </relationshipClass>
            <symbol id="5" class="3" rel="1">
                <boundingBox>
                    <xmin>68</xmin>
```

2.
```
<batch>
  <file type="img">1.bmp</file>
  <file type="img">2.bmp</file>
  <file type="img">3.bmp</file>
  <file type="tex">testset.tex</file>
</batch>
```

3.
```
<expressions>
  <expression>
    <symbol id="56" />
    <symbol id="59" />
    <symbol id="61" />
    <symbol id="57" />
    <symbol id="63" />
    <symbol id="60" />
    <symbol id="58" />
    <symbol id="62" />
  </expression>
  <expression>
    <symbol id="68" />
    <symbol id="64" />
    <symbol id="70" />
    <symbol id="69" />
    <symbol id="65" />
    <symbol id="66" />
    <symbol id="67" />
  </expression>
  <expression>
    <symbol id="73" />
```

4.
```
<div class="equation" style="position:relative;display:inline-bloc
    <div class="symbol" style="position:absolute;top:150px;left:
        <select>
            <option value="0">choose class</option>
            <option value="1">small</option>
            <option value="2">descending</option>
            <option value="3">ascending</option>
            <option value="4">variable range</option>
        </select>
        <div> </div>
    </div>
    <div class="symbol" style="position:absolute;top:94px;left:1
        <select>
            <option value="0">choose class</option>
            <option value="1">small</option>
            <option value="2">descending</option>
            <option value="3">ascending</option>
            <option value="4">variable range</option>
        </select>
        <div> </div>
    </div>
```

Figure A.4: Some XML Files handled by the Proposed System

Additional Buttons. The actions behind the main buttons are self-explanatory. 'Select Parent' allows in fact to create and label a relationship. The child is the currently selected symbol in the image panel. When this button is clicked, the GUI wait for the user to click on the parent symbol and display an instance of `SelectRelationship` for the labelling, as shown on Figure 5.7, page 85.

A.5.2 The Panels of the Main Window

The Image Panel

The implementation of this panel has been explained in Chapter 5. The `ImagePanel` object contains some global variables, which are parameters for the display. For example, to realize what is the actual input of the system, one can choose to display only the bounding boxes. Different kinds of displays are presented on Figure A.5.

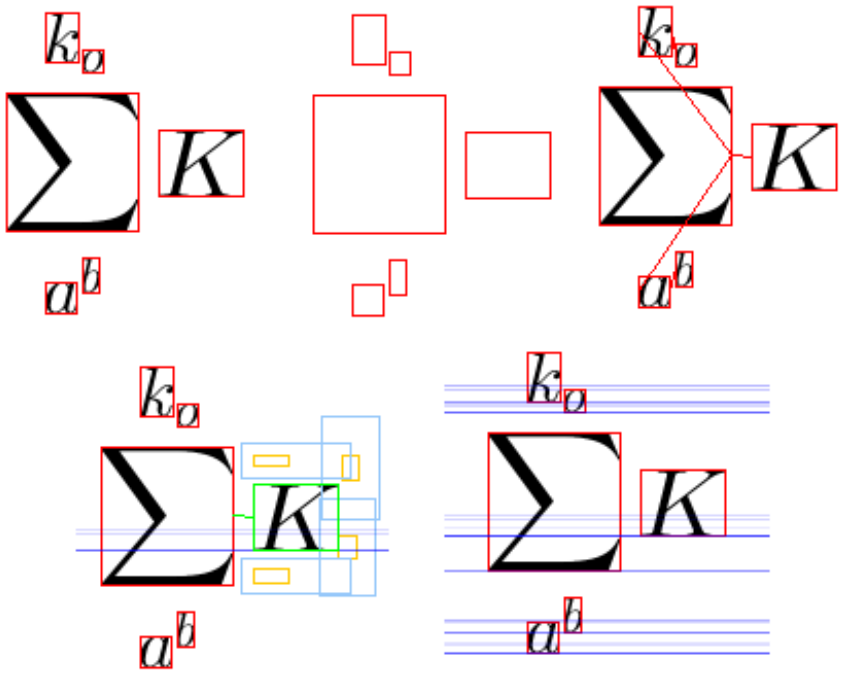

Figure A.5: Image Panel

The Classification Panel

The classification panel, instance of `ClassificationPanel` is used to display the classification of the selected symbol. It uses histograms, which are instances of the class `Histogram`. In the bottom-left corner is the mixed distribution of confidence values for the relationship classifier. On the top line, there are zero to five histograms, for the results of the child based classifiers. On the second line, left to right, are found the results of the ratio-based classifier, of the parent-based classifier and the mix of the children based classifier. On the last line is the mixed result. On figure 5.5, page 84, we notice that the selected symbol has only one child (subscript) and we can see how the system handles the lack of context in the resulting distributions. The `Histogram` object is created from an array of double and a size. It automatically calculates the position and size of the bars.

A.5.3 Implementation of the 'Plot' Window

When an expression is processed by the system, and then compared with the expected interpretation, a `Statistics` object is created. It contains the parameters for the evaluation of the system. It can be seen as a multi-dimensional data point (6-dimensional). When a whole test set is evaluated, we get a vector of `Statistics`. A `PlotWindow` object

is instantiated with such a vector. It provides a two-dimensional projection of the results, making their analysis more convenient. It is made of a top menu, where the features for each axis can be chosen, and of a `PlotPanel` object, where the graph is drawn, with an automatic scaling. An instance of this window is displayed on Figure A.6.

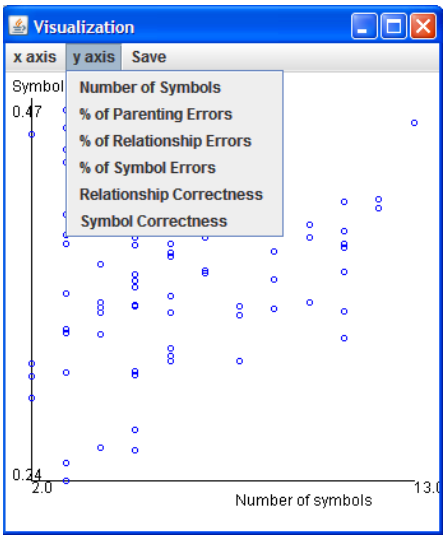

Figure A.6: Plot Window

A.5.4 The File Manager

When a folder is clicked, the workspace moves into it. When a file is clicked, the actions available, given the extension, are shown. For instance,

- 'bmp' files are images which can be opened
- 'tex' files can be parsed to import mathematical expressions
- 'model' files can be read a classifiers, serialized by Weka
- 'xml' can contain list of symbols to load, or expressions interpretations to be imported as expressions or used to test the performance of the system.

The files and folders are represented as `FileButton` objects. The `FileManager` class is a singleton.

A.6 Implementation of Two Algorithms

In this section, we will present how the objects interact with each other to perform what they are meant to do. We show only some algorithms, namely the data set creation and the classification. More algorithm descriptions can be found in the next appendix, and in the API.

A.6.1 Data Set Creation

As explained before, the data set creation is done by a `DatasetBuilder` object. The methods in this object first generate Latex expressions. Then, the class `LatexParser` transforms this expression into an image, which is converted in an `ImageProc` object. The

image is segmented and the `Expression` is created from the list of bounding boxes, and made available to the `DatasetBuilder`. Then, the method `autoProcessDataset(int[], int[])` is called. It allows to use the information about the generation of the expression to automatically label it. The method `getDatasetLine()` of each symbol finally allows to create the CSV file used to populate the database. This can be summarized by the diagram on Figure A.7.

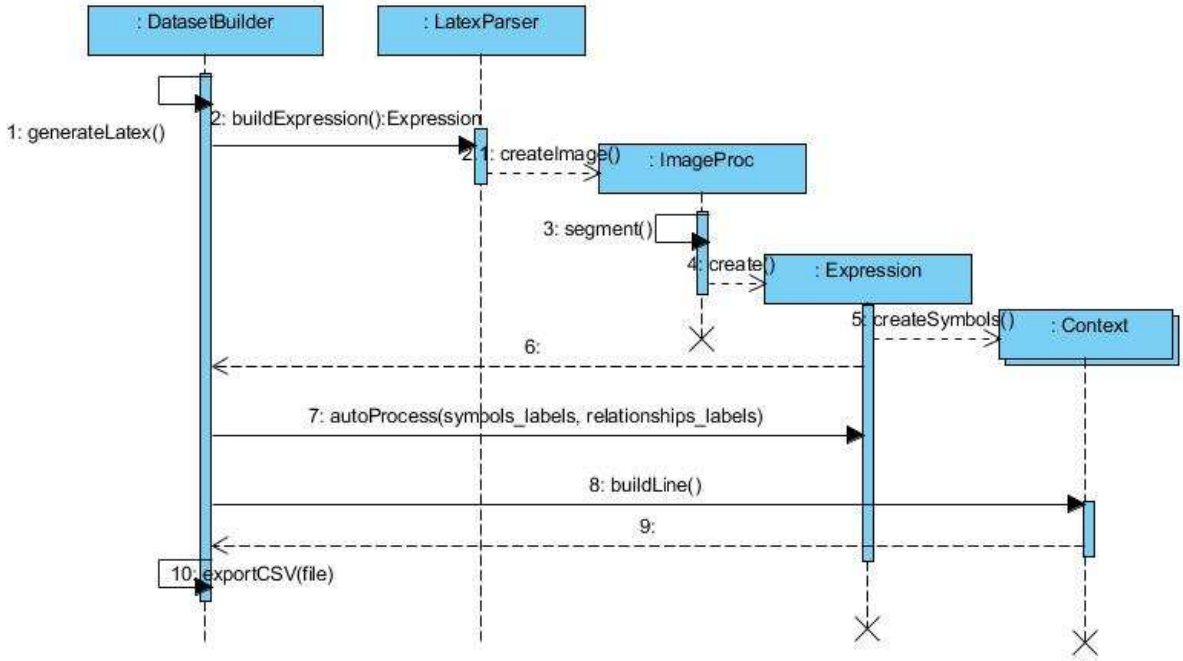

Figure A.7: Sequence Diagram of the Data Set Creation

A.6.2 The Classification

The classification is the core of the problem. It is made of several steps, corresponding to different implementations. The initialization is a rough symbol classification. This is an usual symbol classification, but with no context. Then, at each iteration, all relationships are undone, the structure is recognized, and the symbol classes are refined. This process corresponds to the method `process()` of `Expression`.

A.6.2.1 Symbol Classification

Each symbol in the expression is classified. The inputs of the classifiers are build by the methods in the class `Context`, and send to the object `Classifiers`, which return the confidence values, as explained on Figure A.8.

A.6.2.2 Structure Recognition

To avoid confusion, the previous structure is undone before a new structure recognition. To do so, the parent link of each symbol is cut, as are the children links. The `Relationship` object associated with a symbol is reinitialized. The data structures used in the structure recognition have been presented in Chapter 5. The design of the algorithm has been shown

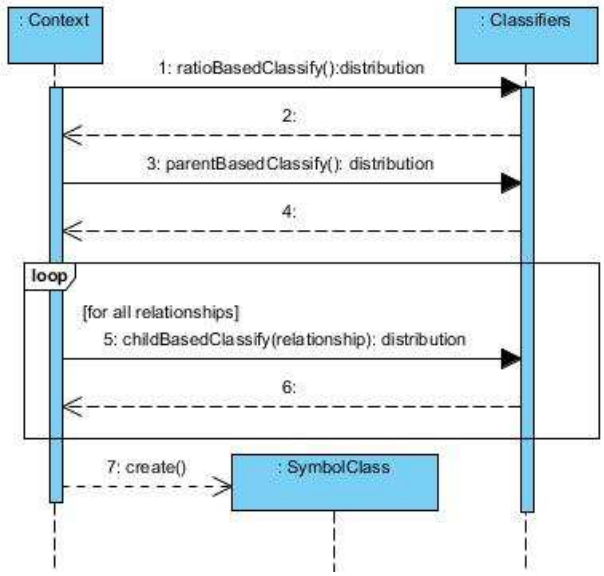

Figure A.8: Sequence Diagram for the Symbol Classification

in Chapter 4. Its implementation is explained by the code in Appendix B. More details can be found in the API on the disk.

Appendix B

Algorithms

In this appendix, we will present the code for several algorithms. They are the main algorithms used to train the classifiers, import inputs, recognize the expressions, test the system and export the results. The whole code and its API is available in the provided disk.

B.1 Creation of the Data Sets

The creation of the data sets is mainly done by the class `DatasetBuilder`. The algorithm presented in Chapter 4 is implemented by methods such as `buildELI()` (presented below, first listing). The automatic generation of instances is performed by methods similar to `buildELILines(int[], int[])`, presented in the second listing. The automatic labelling of these expressions is done within the `Expression` class, via methods such as `autoProcessELIDataset(int[], int[])` (third listing).

Listing B.1: Generation of Expressions

```
1  public void buildELI()
2  {
3    // Operators
4    String[] op = {"", "^{", "_{", "}{"};
5    String eq = "";   // latex expression
6    for (int a=0; a<6; a++)
7      for (int aop=0; aop<3; aop++)
8        for (int b=0; b<6; b++)
9          for (int bop=0; bop<4; bop++)
10           for (int c=0; c<6; c++)
11             if (aop>0 || bop<3)
12             {
13               // for all triplets, twice
14               eq = "";
15               // choose randomly a symbol in the corresponding class
16               eq += symbols[a/2][(int) Math.floor(symbols[a/2].length*Math.random()
                     )];
17               eq += op[aop];
18               eq += symbols[b/2][(int) Math.floor(symbols[b/2].length*Math.random()
                     )];
19               eq += op[bop];
20               eq += symbols[c/2][(int) Math.floor(symbols[c/2].length*Math.random()
                     )];
21               if (aop>0) eq += "}";
22               if (bop>0 && bop<3) eq += "}";
23               // Record classes of symbols and relationships
24               int[] symClasses = {a/2, b/2, c/2};
25               int[] relClasses = {aop, bop};
26
27               ImageProc ip = getImage(eq);       // Create image
```

```
28                Expression expr = new Expression(ip.segment());      // Create
                      expression
29                expr.addGaussian();    // add Gaussian variations
30                buildELILines(expr, symClasses, relClasses); // build examples for NN
31                buildYNCELILines(expr, symClasses, relClasses); // build example for
                      possible-chile classifier
32                total++;
33              }
34  }
```

Listing B.2: Automatic Generation of the Instances

```
1  private void buildELILines(Expression expr, int[] symClasses, int[] relClasses)
2  {
3    if (expr.autoProcessELIDataset(symClasses, relClasses))
4      // Automatic labelling... returns if the segmentation was OK (i.e. 3 symbols)
5      for (Context c: expr.getSymbols())  data[line++] = c.getDatasetLine(); //
           create instances in the dataset
6    else
7      data[line++][1] = 1; // record segmentation error
8  }
```

Listing B.3: Automatic Labelling of Expressions

```
1  public boolean autoProcessELIDataset(int[] symClasses, int[] relClasses) {
2    if (this.symbols.size()!=3) // segmentation error!
3      return false;
4    else
5    {
6      // Automatic parenting
7      this.autoParenting();
8
9      Context c1 = symbols.get(0),
10         c2 = symbols.get(1),
11         c3 = symbols.get(2);
12     // Symbol Labelling
13     c1.setClass(symClasses[0]);
14     c2.setClass(symClasses[1]);
15     c3.setClass(symClasses[2]);
16     // Relationship labelling
17     c2.setRelationship(relClasses[0]);
18     c1.addChild(c2, relClasses[0]);
19     if (relClasses[1]==3)   { c1.addChild(c3, Relationship.INLINE); c3.
           setRelationship(Relationship.INLINE); }
20     else          { c2.addChild(c3, relClasses[1]);  c3.setRelationship(relClasses
           [1]);  }
21
22     return true;
23   }
24 }
```

B.2 Train the Classifiers

The classifiers are trained using the Weak API, in methods of the class `Classifiers`. Here is one example for the parent-based classifier.

Listing B.4: Training the Parent-Based Classifier

```
1
2    String sql;
3    Instances data;
4    Evaluation eval;
5    String[] options;
6
7    // ---
8    // Connect to the database
```

```
9   // ---
10  InstanceQuery query = new InstanceQuery();
11  query.setDatabaseURL(dbase);
12  query.setUsername("");
13  query.setPassword("");
14
15  // ---
16  // Set query
17  // ---
18
19  sql = "SELECT ";
20  sql += "Data.H, Data.D, Data.V, ";
21  sql += "Data.PARENT_CHAR AS PCLASS, ";
22  sql += "Data.CLASS ";
23  sql += "FROM Data ";
24  sql += "WHERE (((Data.SEGERR)=0) AND (Data.PARENT_CHAR<>'0') );";
25
26  query.setQuery(sql);
27  data = query.retrieveInstances();
28
29  // ---
30  // Setting options
31  // ---
32  options = Utils.splitOptions("-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a");
33  SCB.setOptions(options);
34  data.setClassIndex(data.numAttributes()-1);
35
36  // ---
37  // Train the classifier
38  // ---
39  System.out.println("Building SCB ...");
40  SCB.buildClassifier(data);
41  System.out.println("Done.");
42
43  // ---
44  // Classifier evaluation
45  // ---
46  System.out.println("Cross-validation for SCB...");
47  eval = new Evaluation(data);
48  eval.crossValidateModel(SCB, data, 10, new Random(1));
49  System.out.println("Done.");
50  System.out.println(eval.toSummaryString("\n Results for SCB: \n\n", false));
```

B.3 Segmentation

The segmentation consists in returning a list of bounding boxes from an input image. It is implemented by the method `segment()` of the class `ImageProc`.

Listing B.5: Image Segmentation

```
1  public Vector<Context> segment ()
2  {
3    // Size of the arrays
4    int max_labels = 100;
5
6    // Initializations
7    int[][] labels = new int[this.width()][this.height()];  // pixel labels
8    int[]   equiv  = new int[max_labels];          // equivalence table
9    boolean[] used = new boolean[max_labels];       // used labels
10   int[][] bbs    = new int[max_labels][4];        // Bounding boxes
11
12   Vector<Context> result = new Vector<Context>();     // Result of the segm.
13
14   // Initialization of the equivalence table
15   for (int i=0; i<max_labels; i++)
16   {
17     equiv[i] = i;   // all labels are equivalent to themselves
```

```
18      used[i] = false; // none are used
19    }
20
21    int current_label = 0;
22    int[] neigh;
23    int nb_nei;
24    int min_label;
25
26    // First Pass
27    for (int y=1; y<this.height()-1; y++)
28      for (int x=1; x<this.width()-1; x++)
29      {
30        // not white pixels
31        if (this.getGrey(x, y)<250)
32        {
33          neigh  = new int[4];
34          nb_nei = 0;
35
36          // Collect neighbours
37          if (labels[x-1][y]>0)   neigh[nb_nei++] = labels[x-1][y];
38          if (labels[x-1][y-1]>0) neigh[nb_nei++] = labels[x-1][y-1];
39          if (labels[x][y-1]>0)   neigh[nb_nei++] = labels[x][y-1];
40          if (labels[x+1][y-1]>0) neigh[nb_nei++] = labels[x+1][y-1];
41
42          if (nb_nei==0)
43            // new label if no neighbour
44            labels[x][y] = ++current_label;
45          else
46          {
47            min_label = max_labels;
48            // get the minimum label
49            for (int i=0; i<nb_nei; i++)
50              if (neigh[i]<min_label) min_label = equiv[neigh[i]];
51            // set pixel label
52            labels[x][y] = equiv[min_label];
53            // update equivalence table
54            for (int i=0; i<nb_nei; i++)
55              if (equiv[neigh[i]]>min_label) equiv[neigh[i]]=equiv[min_label];
56          }
57        }
58      }
59
60    // update equivalence table to get the min equivalent
61    for (int i=0; i<max_labels; i++)
62      while (equiv[i] > equiv[equiv[i]]) equiv[i] = equiv[equiv[i]];
63
64    // Second pass
65    int actLabel;
66    for (int x=1; x<this.width(); x++)
67      for (int y=1; y<this.height()-1; y++)
68      {
69        if (labels[x][y]>0)
70        {
71          // If the pixel is labelled
72          actLabel = equiv[labels[x][y]];
73          if (!used[actLabel])
74          {
75            // initialize bounding box
76            bbs[actLabel][0] = x;
77            bbs[actLabel][1] = x;
78            bbs[actLabel][2] = y;
79            bbs[actLabel][3] = y;
80            used[actLabel] = true;
81          }
82          else
83          {
84            // update bounding box
85            if (bbs[actLabel][0] > x) bbs[actLabel][0] = x;
86            if (bbs[actLabel][1] < x) bbs[actLabel][1] = x;
87            if (bbs[actLabel][2] > y) bbs[actLabel][2] = y;
```

```
88            if (bbs[actLabel][3] < y) bbs[actLabel][3] = y;
89          }
90        }
91      }
92
93    // Create Context objects corresponding to the bounding boxes
94    for (int i=0; i<max_labels; i++)
95      if (used[i])
96      {
97        result.add(new Context(bbs[i]));
98      }
99
100   return result;
101 }
```

B.4 Parse Latex File

The `LatexParser` object allows to open a Tex file and look for equations in it. It is a convenient way of storing a set of expressions. The parsing result is a list of Latex equations. Afterwards, the method `getLatexImage(String)` can create the binary image corresponding to these equations.

Listing B.6: Latex File Parsing

```
1  public LatexParser(String path) throws FileNotFoundException
2  {
3    // Opens the file
4    BufferedReader in  = new BufferedReader(new FileReader(path));
5    // Initialize the vector of expressions
6    expressions = new Vector<String>();
7    String line;
8
9      try
10   {
11     // Read the first line
12     line = in.readLine();
13     Pattern pattern = Pattern.compile("&begin\\{equation\\}(.*)&end\\{equation\\}")
           ;
14     Matcher matcher;
15     // browse the document
16     while(line!=null)
17     {
18       // rewrite the line
19       line = line.replace("\\begin", "&begin").replace("\\end", "&end");
20       matcher = pattern.matcher(line);
21         if(matcher.matches())
22         {
23           // add the Latex equation
24           expressions.add(matcher.group(1));
25         }
26         line = in.readLine(); //next line
27     }
28   } catch (IOException e) { e.printStackTrace(); }
29 }
```

B.5 Symbol Classification

When `classifySymbols()` in `Expression` is called, the method `classifySymbol()` of each `Context` is called. It uses the method `classifySymbol(Context)` of the `Classifiers` object, presented below. It results in a `SymbolClass` object which mixes the classifiers outputs and represents the classification, which can be associated with the symbol.

Listing B.7: Symbol Classification

```
1  public SymbolClass classifySymbol(Context c) throws Exception
2  {
3    // Get context information
4    boolean hasParent = c.hasParent(), hasSub = c.hasSub(), hasSup = c.hasSup(),
         hasHor = c.hasHor(), hasUpp = c.hasUpp(), hasUnd = c.hasUnd();
5    // Define arrays
6    double[] distA, distB, distC1, distC2, distC3, distC4, distC5;
7
8    distA  = getSymbolClassA(c); // Ratio-based classification
9    // Divide by prior on symbol class
10   for (int i=0; i<Parameters.NB_OF_SYMBOL_CLASSES; i++) distA[i] /= SymbolClass.
         PROPCL[i];
11   Utils.normalize(distA); // Normalize array
12   // Parent-based classification
13   distB  = (hasParent) ? getSymbolClassB(c) : ArrayTools.evenDist(Parameters.
         NB_OF_SYMBOL_CLASSES);
14   // Children-based classifications
15   distC1 = (hasHor) ? getSymbolClassC(c,Relationship.INLINE)      : ArrayTools.
         evenDist(Parameters.NB_OF_SYMBOL_CLASSES);
16   distC2 = (hasSup) ? getSymbolClassC(c,Relationship.SUPERSCRIPT) : ArrayTools.
         evenDist(Parameters.NB_OF_SYMBOL_CLASSES);
17   distC3 = (hasSub) ? getSymbolClassC(c,Relationship.SUBSCRIPT)   : ArrayTools.
         evenDist(Parameters.NB_OF_SYMBOL_CLASSES);
18   distC4 = (hasUpp) ? getSymbolClassC(c, Relationship.UPPER)      : ArrayTools.
         evenDist(Parameters.NB_OF_SYMBOL_CLASSES);
19   distC5 = (hasUnd) ? getSymbolClassC(c, Relationship.UNDER)      : ArrayTools.
         evenDist(Parameters.NB_OF_SYMBOL_CLASSES);
20   // Create symbol class (SymbolClass will mix the confidence values)
21   return new SymbolClass(distC1, distC2, distC3, distC4, distC5, distA, distB);
22 }
```

B.6 Structure Recognition

The structure recognition is performed by the method `parentAndRelate()` af a `RelationshipFinder` object, as presented below.

Listing B.8: Structure Recognition

```
1  public void parentAndRelate()
2  {
3    // Initialize the stack of seen symbols
4    Stack<BaselineStructure> lastBL = new Stack<BaselineStructure>();
5    // Retrieve the list of symbols
6    Vector<Context> vc = expr_.getSymbols();
7    // Get the first symbol...
8    Context considered = expr_.findLeftMost();
9    // ... which has no parent: we add it to the stack of seen symbols
10   BaselineStructure dominantBLS = new BaselineStructure(considered);
11   lastBL.add(dominantBLS);
12
13   // The working stack
14   Stack<BaselineStructure> lastBL_clone;
15   // The candidate baseline
16   BaselineStructure consideredBL;
17   // The list of candidate baselines
18   Vector<BaselineStructure> candidatesBL;
19   // The candidate for parenting, and the best parent found so far
20   Context possibleParent; BaselineStructure bestParent;
21   // Has a parent been found
22   boolean parented;
23   // The variables for the confidence of a relationship
24   double confidence, max_confidence;
25   // The considered relationship, and the best found so far
26   int relation, bestRelation;
27   // The confidence values for relationship (RC)
28   double[] rc;
```

```
29    // The confidence values for relationship (fuzzy regions)
30    double[] fuzzy;
31    // The baseline confidence
32    double[] baseline;
33    // The YNC confidence
34    double[] possibleChild;
35    // The array containing the relationship sorted by confidence
36    int[] sortedRels;
37
38    // We browse the symbols in the list
39    for (int i=1; i<vc.size(); i++)
40    {
41      // Retrieve the next symbol
42      considered = vc.get(i);
43      // Not yet parented
44      parented = false;
45      // Default best parent
46      bestParent = dominantBLS;
47      // Default best relation
48      bestRelation = Relationship.INLINE;
49
50      max_confidence = 0.;
51
52      // ------------------
53      // BEGIN INLINE TEST
54
55      // Start with the dominant baseline
56      candidatesBL = new Vector<BaselineStructure>();
57      candidatesBL.add(dominantBLS);
58
59      while (!parented && candidatesBL.size()>0)
60      {
61        // Check all baselines, beginning with the dominant
62        consideredBL = candidatesBL.remove(0);
63        possibleParent = consideredBL.getDominantSymbol();
64        relation = Relationship.INLINE;    // inline test
65        confidence = 0.;
66
67        // Retrieve the confidence values
68        try {
69          rc = considered.getVirtualRelationship(possibleParent);
70          fuzzy = possibleParent.getFuzzyRegions().memberships(considered);
71          baseline = considered.baselineScore(possibleParent);
72          possibleChild = considered.possibleChildOf(possibleParent);
73        } catch(Exception e) {
74          rc = new double[Parameters.NB_OF_RELATIONSHIP_CLASSES];
75          fuzzy = new double[Parameters.NB_OF_RELATIONSHIP_CLASSES];
76          baseline = new double[Parameters.NB_OF_SYMBOL_CLASSES];
77          possibleChild = new double[2];
78          MainWindow.inst.addResultText("Error in the classification");
79        }
80        // Mix the confidence to have the globa confidence
81        confidence = rc[Relationship.INLINE]*Parameters.COEFF_RCI;
82        confidence += baseline[considered.getSymbolClass()-1]*Parameters.COEFF_BL;
83        confidence += possibleChild[0]*Parameters.COEFF_YNI;
84        confidence /= (Parameters.COEFF_BL+Parameters.COEFF_RCI+Parameters.COEFF_YNI)
              ;
85
86        // If the confidence is high enough, we can do the parenting for this
              relationship
87        if (confidence > THRESHOLD)
88        {
89          parented = true;
90          considered.setRelationship( new Relationship(possibleParent, considered,
                relation) );
91          consideredBL.updateStructure(considered); // the baseline structure is
                updated
92          lastBL.push(consideredBL); // symbol added to the stack
93        }
94        // If not, it can be the best parent amongst all symbols
```

```
95        else
96        {
97          if (confidence > max_confidence)
98          {
99            max_confidence = confidence;
100           bestParent = consideredBL;
101           bestRelation = relation;
102         }
103         // Retrieve nested baselines for the next step in inline test
104         Vector<BaselineStructure> nested = consideredBL.getNestedBaselines();
105         if (nested!=null) candidatesBL.addAll(nested);
106       }
107     }
108     // END OF THE INLINE TEST
109     // ----------------------
110     if (!parented)
111     {
112       // ----------------
113       // BEGIN CHILD TEST
114       lastBL_clone = (Stack<BaselineStructure>) lastBL.clone();
115       do
116       {
117         consideredBL = lastBL_clone.pop();  // Get the last symbol (1/2)
118         possibleParent = consideredBL.getDominantSymbol(); // (2/2)
119         // Retrieve the confidence values
120         try {
121           rc = considered.getVirtualRelationship(possibleParent);
122           fuzzy = possibleParent.getFuzzyRegions().memberships(considered);
123           baseline = considered.baselineScore(possibleParent);
124           possibleChild = considered.possibleChildOf(possibleParent);
125         } catch(Exception e) {
126           rc = new double[Parameters.NB_OF_RELATIONSHIP_CLASSES];
127           fuzzy = new double[Parameters.NB_OF_RELATIONSHIP_CLASSES];
128           baseline = new double[Parameters.NB_OF_SYMBOL_CLASSES];
129           possibleChild = new double[2];
130           MainWindow.inst.addResultText("Error in the classification - 2");
131         }
132         // -- Loop on relationships
133         // Sort the RC relationships
134         sortedRels = ArrayTools.indexSort(rc);
135         int r=0;
136         // Keep looking for a possible relationship while it is not parented,
137         //   in order of the most likely relationships
138         while (!parented && r<Parameters.NB_OF_RELATIONSHIP_CLASSES)
139         {
140           relation = sortedRels[r];   // Get the considered relationship
141           confidence = Parameters.COEFF_RCC*rc[relation];   // Get the confidence
                for it
142           if (!possibleParent.hasChild(relation) && relation!=Relationship.INLINE)
143           {
144             // Mix the confidence if this relationship is available
145             confidence += Parameters.COEFF_FR*fuzzy[relation];
146             confidence += Parameters.COEFF_YNC*possibleChild[1];
147             confidence /= (Parameters.COEFF_BL+Parameters.COEFF_RCC+Parameters.
                  COEFF_YNC);
148             // If the confidence is high enough, we can do the parenting for this
                  relationship
149             if (confidence > THRESHOLD)
150             {
151               parented = true;
152               considered.setRelationship( new Relationship(possibleParent,
                    considered, relation) );
153               BaselineStructure newNode = new BaselineStructure(considered);
154               consideredBL.addChild(newNode);
155               lastBL.push(newNode);
156             }
157             // If not, it can be the best parent amongst all symbols
158             else
159             {
160               if (confidence > max_confidence)
```

```
161                 {
162                   max_confidence = confidence;
163                   bestParent = consideredBL;
164                   bestRelation = relation;
165                 }
166               }
167             } // end if no parent
168             r++;
169           } // end loop on relationships
170         } while(!parented && !consideredBL.equals(dominantBLS));
171         // END OF CHILD TEST
172         // ------------------
173       }
174       if (!parented)
175       {
176         // Parent with the best parent, if possible (ie if no child in that rel.)
177         possibleParent = bestParent.getDominantSymbol();
178         BaselineStructure newNode = new BaselineStructure(considered);
179         if (!possibleParent.hasChild(bestRelation))
180         {
181           considered.setRelationship( new Relationship(possibleParent, considered,
                bestRelation) );
182           if (bestRelation==Relationship.INLINE)
183             bestParent.updateStructure(considered);
184           else
185             bestParent.addChild(newNode);
186         }
187         lastBL.push(newNode);
188       }
189     } // next symbol
190   }
```

B.7 Iterative Algorithm

To recognize an expression, the method `process()` of the `Expression` is called. It returns a list of `Interpretation` objects, corresponding to the expression's interpretations at each step.

Listing B.9: An Iterative Algorithm

```
1  public Vector<Interpretation> process()
2  {
3    interpretations = new Vector<Interpretation>();
4    RelationshipFinder rf = new RelationshipFinder(this);
5    this.classifySymbols();    // Rough symbols classification
6    for (int iter=0; iter<Parameters.ITERATIONS; iter++)
7    {
8      this.clearRelationships();       // undo relationships
9      rf.parentAndRelate();        // structure recognition
10     this.addCurrentInterpretation();  // save interpretation
11     this.classifySymbols();        // symbols classification
12     this.addCurrentInterpretation();  // save interpretation
13   }
14   return interpretations;
15 }
```

B.8 Export XML Interpretation

`XMLCreator` provides several methods to create and read XML files. To create the XML interpretation of an expression, the method `createXML(Expression, String)` is called. It creates the root node (`<expression width=.. height=..>`) which contains one child, the dominant symbol node. The JDOM framework is used to create and save the XML file, and the symbol nodes are created by `createSymbolNode(Context)`.

Listing B.10: Export XML Interpretation

```
1  public static Element createSymbolNode(Context c)
2  {
3    Element e;
4    double[] d;
5
6    // Create the node
7    Element el = new Element("symbol");
8    // Set the attributes
9    el.setAttribute(new Attribute("id",     ""+c.getSymbolId()));
10   el.setAttribute(new Attribute("class", c.getSymbolClass()+""));
11   el.setAttribute(new Attribute("rel",    c.getRelationship()+""));
12
13   // Child: bounding box
14   e = new Element("boundingBox");
15   e.addContent( new Element("xmin").addContent(c.getSymbol().xmin+"") );
16   e.addContent( new Element("xmax").addContent(c.getSymbol().xmax+"") );
17   e.addContent( new Element("ymin").addContent(c.getSymbol().ymin+"") );
18   e.addContent( new Element("ymax").addContent(c.getSymbol().ymax+"") );
19
20   el.addContent(e);
21
22   // Child: confidence on the symbol class
23   e = new Element("symbolClass");
24   d = c.getSymbolClassObject().confidences();
25   for (int i=0; i<Parameters.NB_OF_SYMBOL_CLASSES; i++)
26     e.addContent( new Element("class").setAttribute("id", (i+1)+"").addContent(d[i
         ]+"") );
27
28   el.addContent(e);
29
30   // Child: confidence on the relationship
31   e = new Element("relationshipClass");
32   d = c.getRelationshipObject().confidences();
33   for (int i=0; i<Parameters.NB_OF_RELATIONSHIP_CLASSES; i++)
34     e.addContent( new Element("class").setAttribute("id", i+"").addContent(d[i]+"")
         );
35
36   el.addContent(e);
37
38   // Add children symbols
39   if (c.hasSup()) el.addContent(createSymbolNode(c.getSup()));
40   if (c.hasSub()) el.addContent(createSymbolNode(c.getSub()));
41   if (c.hasUpp()) el.addContent(createSymbolNode(c.getUpp()));
42   if (c.hasUnd()) el.addContent(createSymbolNode(c.getUnd()));
43   if (c.hasHor()) el.addContent(createSymbolNode(c.getHor()));
44
45   return el;
46 }
```

B.9 Compare Expressions

An expression can be compared to a reference (for example, extracted from an XML file), via the method `compare(Expression)` of the class `Expression`. It returns a list of statistics concerning the comparison, in the form of a `Statistics` object.

Listing B.11: Compare Two Expressions

```
1  public Statistics compare(Expression e)
2  {
3    // First requirement: same number of symbols
4    boolean eq = this.getSymbols().size() == e.getSymbols().size();
5    if (!eq) MainWindow.inst.addResultText("The expressions do not have the same
         amount of symbols!\n (this:"+this.nbSymbols()+", e:"+e.nbSymbols()+")\n");
6    // Initialize the iterators
7    Iterator<Context> iterThis  = this.getSymbols().iterator();
```

```
8   Iterator<Context> iterOther = e.getSymbols().iterator();
9   // Initialize Statistics parameters
10  int n = this.getSymbols().size();
11  double[] correctnessSymbol = new double[n];
12  double[] correctnessRelationship = new double[n];
13  int symErr=0,parErr=0,relErr=0;
14  int[] symErrClass = new int[Parameters.NB_OF_SYMBOL_CLASSES];
15  int[] strucErr = new int[5];
16  int[] errByCtxt = new int[Parameters.NB_OF_RELATIONSHIP_CLASSES+2];
17  int[][] symConf = new int[Parameters.NB_OF_SYMBOL_CLASSES][Parameters.
        NB_OF_SYMBOL_CLASSES];
18  // Iteration on symbols if possible
19  int i = 0;
20  boolean parentingErrorFound;
21  while (eq && iterThis.hasNext())
22  {
23    Context c1 = iterThis.next(); // next in this
24    Context c2 = iterOther.next();  // next in reference
25    // Initialize parenting error
26    parentingErrorFound = false;
27    // Update confusion matrix
28    symConf[c2.getSymbolClass()-1][c1.getSymbolClass()-1]++;
29
30    if (c1.getSymbolClass()!=c2.getSymbolClass())   // symbol misclassification?
31    {
32      symErr++;                 // update stats
33      symErrClass[c2.getSymbolClass()-1]++;
34      errByCtxt[c2.amountOfContext()]++;
35    }
36    if (c1.getRelationship()!=c2.getRelationship()) // relationship error?
37    {
38      relErr++; // update stats
39      if (c2.hasParent()) // Check if it corresponds to an misclassification of
            parent...
40        if (this.getContextById(c2.getParent().getSymbolId()).getSymbolClass()!=c2.
              getParent().getSymbolClass()) strucErr[3]++;
41      if (c1.getSymbolClass()!=c2.getSymbolClass())  strucErr[4]++; // ... and/or
            of child
42    }
43    // Check if there is a parenting error
44    if (c1.hasParent())
45    {
46      if (!c2.hasParent()) // parent in expression, none in reference
47      {
48        parErr++;
49        parentingErrorFound=true;
50        strucErr[0]++;
51      }
52      else if (c1.getParent().getSymbolId()!=c2.getParent().getSymbolId())
53      { // not the same parent
54        parErr++;
55        parentingErrorFound=true;
56        if (this.getContextById(c2.getParent().getSymbolId()).getSymbolClass()!=c2.
              getParent().getSymbolClass()) strucErr[0]++;
57      }
58    }
59    else if (c2.hasParent())
60    { // parent in reference, none in this expression
61      parErr++;
62      parentingErrorFound=true;
63      if (this.getContextById(c2.getParent().getSymbolId()).getSymbolClass()!=c2.
            getParent().getSymbolClass()) strucErr[0]++;
64    }
65    // Compute corretness (confidence of the actual class)
66    correctnessSymbol[i] = c1.getSCConfidence(c2.getSymbolClass());
67    if (parentingErrorFound)
68    { // if error parenting, the rel. correctness correspond to the conf. of actual
          rel. and parent
69      correctnessRelationship[i] = c1.getVirtualRelConfidence(c2.getParent(), c2.
            getRelationship());
```

```
70        if (c1.getRelationship()!=c2.getRelationship()) strucErr[2]++; // parenting
              err. due to rel. error
71        if (c1.getSymbolClass()!=c2.getSymbolClass())  strucErr[1]++;  // ... or symb
              . misclassification
72      }
73      else // relationship correctness
74        correctnessRelationship[i] = c1.getRelConfidence(c2.getRelationship());
75      // next
76      i++;
77    }
78    // create Statistics object
79    Statistics stat = new Statistics();
80    stat.setNbSymbols(n);
81    stat.setErrorParenting(parErr); stat.setErrorSymbol(symErr); stat.
          setErrorRelationship(relErr);
82    stat.setErrorSymbolByClass(symErrClass); stat.setErrorSymbolByContext(errByCtxt);
83    stat.setConfusionMatSym(symConf); stat.setStructureError(strucErr);
84    stat.setCorrectnessRelationship(correctnessRelationship); stat.
          setCorrectnessSymbol(correctnessSymbol);
85    stat.setExpressionLatex(this.getLatex()); stat.setFoundLatex(this.buildMetaLatex
          ()); stat.setExpectedLatex(e.buildMetaLatex());
86
87    return stat;
88  }
```

Appendix C

Test Sets and Results

The aim of this appendix is to provide a comprehensive presentation of the results. We will present the test sets, and show the results yielded by the system, as well as some statistics.

C.1 The Test Sets

All test sets have been built using the developed GUI. They have been designed and labelled by hand. We loaded a set of LaTeXexpressions from a TeX file that we wrote. Using the tools in the GUI, we labelled the expressions, and exported the interpretations, which correspond to this labelling. Then, we could reload the expressions, and apply the recognition to them. Finally, we could compare them with the saved interpretations.

We have different test sets, for different complexities of expressions and different content. We included in the test sets some symbols which were not in the training sets, in order to evaluate the flexibility of the system.

We also have a test set of expressions generated by other equations editor than LaTeX, and a test set of handwritten expressions. The aim is to see how flexible our system is. Indeed, it has been built using only expressions generated by the *jLaTeXMath* framework.

C.2 Notations and Figures

We can see how the system works, and understand some mistakes by looking at the recognition output. However, showing the uncertainty and confidence values for each symbol is difficult. In the next section, we will show only the most likely interpretation, along with some information such as the average confidence on the actual symbols' class (symbol correctness).

To show a visual output, we generate a meta-Latex expression. Indeed, since we do not perform a symbol recognition but a symbol classification, we cannot show the symbols' identity. Thus, in the results, 'a' represents a symbol of class 'small', 'p' for 'descending', 'b' for 'ascending' and '$\sum$' for 'variable range'. To avoid confusion, we also present the expected meta-Latex expression, which is merely the translation of the actual Latex expression into its meta-Latex form.

A summary of the recognition is also presented in the last column. N is the number of symbols, e_r the ratio of relationships errors, e_p the ratio of parenting errors, e_s the ratio of symbol misclassifications, C_s the average symbol correctness and C_r the average relationship correctness. These last two numbers represent how good the system is. It uses the uncertainty that is left in the interpretation. Global results have been presented in the chapter 'Results and Evaluation'.

C.3 Results

The following tables present the results of each test set. They show the actual expression, the expected Latex output and the actual one, and the recognition statistics. Some recognition results cannot be interpreted by Latex, when there is a subscript and an under symbol for example. The plain Latex command in then presented instead.

The expressions marked with (*) have been used for human labelling.

Table C.1: Results for testset0-1 (left) and testset0-2 (right)

Actual Expression	Expected Result	Recognition Result	Statistics
nocontext	*aaaaaabaab*	*aaaaabaab*	$N = 9$, $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.41$, $C_r = 0.64$
dc	*ba*	*pa*	$N = 2$, $e_s = 50\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.3$, $C_r = 0.73$
vlm	*aba*	*aba*	$N = 3$, $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.44$, $C_r = 0.2$
EPx	*bba*	*bba*	$N = 3$, $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.39$, $C_r = 0.5$
123	*bbb*	*bbb*	$N = 3$, $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.47$, $C_r = 1.0$
1a2b3p	*babbbp*	*babbbp*	$N = 6$, $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.46$, $C_r = 0.67$
tanπ	*baaa*	*baaa*	$N = 4$, $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.44$, $C_r = 0.61$
$\sum^{a}$	$\sum a$	b_a	$N = 2$, $e_s = 50\%$, $e_p = 0\%$, $e_r = 50\%$, $C_s = 0.29$, $C_r = 0.0$
$\sum pacb$	$\sum paab$	$a_{b^a \Sigma b}$	$N = 5$, $e_s = 60\%$, $e_p = 20\%$, $e_r = 40\%$, $C_s = 0.3$, $C_r = 0.44$
$\prod_{a} \bigcup_{N}$	$\sum_{a} \sum b$	$\sum ap_a$	$N = 4$, $e_s = 50\%$, $e_p = 0\%$, $e_r = 25\%$, $C_s = 0.34$, $C_r = 0.45$

Actual Expression	Expected Result	Recognition Result	Statistics
$\bigcap \bigcup Y$	$\sum \sum b$	pp_a	$N = 3$, $e_s = 100\%$, $e_p = 0\%$, $e_r = 33\%$, $C_s = 0.25$, $C_r = 0.5$
$\prod^{25nB}$	$\sum bbab$	$\sum bbaa$	$N = 5$, $e_s = 20\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.43$, $C_r = 0.75$
$5 \sum gh$	$b \sum pb$	ba_{p^p}	$N = 4$, $e_s = 50\%$, $e_p = 0\%$, $e_r = 50\%$, $C_s = 0.26$, $C_r = 0.55$
$a \sum n \prod wp$	$a \sum a \sum ap$	$ab_a \sum ap$	$N = 6$, $e_s = 16\%$, $e_p = 16\%$, $e_r = 16\%$, $C_s = 0.35$, $C_r = 0.57$
$PadV$	$babb$	$ba^p a$	$N = 4$, $e_s = 50\%$, $e_p = 25\%$, $e_r = 25\%$, $C_s = 0.35$, $C_r = 0.45$
$PV\ nRT$	$bbabb$	$bbaaa$	$N = 5$, $e_s = 40\%$, $e_p = 0\%$, $e_r = 0\%$, $C_s = 0.36$, $C_r = 0.72$
$\alpha \Gamma \pi \Delta$	$abab$	$b^{\sum} aa$	$N = 4$, $e_s = 75\%$, $e_p = 25\%$, $e_r = 25\%$, $C_s = 0.33$, $C_r = 0.5$
$\bigwedge \alpha p$	$\sum ap$	p_{ap}	$N = 3$, $e_s = 33\%$, $e_p = 0\%$, $e_r = 33\%$, $C_s = 0.35$, $C_r = 0.64$
$\sigma \bigcup Au$	$a \sum ba$	ap_{aa}	$N = 4$, $e_s = 50\%$, $e_p = 0\%$, $e_r = 25\%$, $C_s = 0.34$, $C_r = 0.2$
$*\varphi \bigvee \bigwedge T p$	$p \sum \sum bp$	$\sum pp_{ap}$	$N = 5$, $e_s = 80\%$, $e_p = 0\%$, $e_r = 20\%$, $C_s = 0.27$, $C_r = 0.54$

Table C.2: Results for testset1-1 (left) and testset1-2 (right)

Actual Expression	Expected Result	Recognition Result	Statistics
a^{b}	a^{b}	a^{b}	$N = 2$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.45$, $C_r = 0.99$
d_c	b_a	p_a	$N = 2$ $e_s = 50\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.31$, $C_r = 0.99$
$b_a\, p^{n}$	$b_a p^{a}$	$b_a p^{a}$	$N = 4$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.45$, $C_r = 0.99$
t_p^m	b_p^a	b_p^a	$N = 3$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.43$, $C_r = 0.99$
$*v_u^A p_a$	$a_a^b p_a$	$a_a^b p_a$	$N = 5$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.43$, $C_r = 0.87$
$a_0 h_u^n$	$a_b a_a^a$	$a_b a_a^a$	$N = 5$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.43$, $C_r = 0.98$
$y_l^n z q_p$	$p_b^a a p_p$	$p_b^a a p_p$	$N = 6$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.41$, $C_r = 0.9$
$b^e pe$	$b^a pa$	$b^a p_a$	$N = 4$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 25\%$ $C_s = 0.39$, $C_r = 0.66$
$\sum_0^n a$	$\sum_b^a a$	$\sum_b^a a$	$N = 4$ $e_s = 0\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.41$, $C_r = 0.94$
$\prod_p^q Y_n$	$\sum_p^p b_a$	$\sum_p^p a_a$	$N = 5$ $e_s = 20\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.39$, $C_r = 0.97$

Actual Expression	Expected Result	Recognition Result	Statistics
$cos^2\theta$	$aaa^{b}b$	$\sum aa^{b}b$	$N = 5$ $e_s = 20\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.4$, $C_r = 0.96$
$l_o^t so_v c_n^e t^e x_t$	$b_a^b aa_a a_a^a b^a a_b$	$b_{a\sum}^b a_a a_a^a b^a a_b$	$N = 13$ $e_s = 7\%$, $e_p = 15\%$, $e_r = 7\%$ $C_s = 0.46$, $C_r = 0.98$
$*\sum_v^g \prod_0^q t_u r^h$	$\sum_a^p \sum_b^p b_a a^b$	$\sum_a^p \sum_b^p b_a a^p$	$N = 10$ $e_s = 10\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.4$, $C_r = 0.95$
$log_2 x^r$	$bap_b a^a$	bap_{ba^a}	$N = 6$ $e_s = 0\%$, $e_p = 16\%$, $e_r = 0\%$ $C_s = 0.43$, $C_r = 0.76$
$\bigcup_H^T h_m^n \bigcap_t x_b$	$\sum_b^b b_a^a \sum_b a_b$	p_{b^{a}_{a} a_{b}}H \sum^{a} a_{b} _{a b}	$N = 10$ $e_s = 30\%$, $e_p = 10\%$, $e_r = 20\%$ $C_s = 0.39$, $C_r = 0.9$
$2\sum_0^Z x\ 42 \prod_v^b y_n$	$b\sum_b^b abb \sum_a^a p_a$	b b_{a b b} ^{a}_{b} \sum_{a} p_{a}	$N = 11$ $e_s = 18\%$, $e_p = 9\%$, $e_r = 9\%$ $C_s = 0.38$, $C_r = 0.82$
$\Gamma_\pi^\alpha \Delta$	$b_a^a b$	$b_a^a a$	$N = 4$ $e_s = 25\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.37$, $C_r = 0.99$
$\bigoplus_s^k \delta_x$	$\sum_a^b b_a$	$\sum_a^p b_a$	$N = 5$ $e_s = 20\%$, $e_p = 0\%$, $e_r = 0\%$ $C_s = 0.36$, $C_r = 0.96$
$\psi \bigotimes_w \prod k^p$	$p\sum_a \sum b^p$	$pa_{a\sum p^p}$	$N = 6$ $e_s = 33\%$, $e_p = 16\%$, $e_r = 0\%$ $C_s = 0.32$, $C_r = 0.94$
$not \bigwedge_s^U \bigvee_q pq^2$	$aab\sum_a^b \sum_p pp^b$	a a b \sum^{a}_{a} p_{p p^{b}}_{p}C	$N = 11$ $e_s = 18\%$, $e_p = 0\%$, $e_r = 9\%$ $C_s = 0.37$, $C_r = 0.8$

Table C.3: Results for testset2-1 (left) and testset2-2 (right)

Actual Expression	Expected Result	Recognition Result	Statistics
5^{23}	b^{bb}	b^{bb}	$N = 3$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.46,\ C_r = 0.99$
$ta^{paq}hg$	$ba^{pap}bp$	$ba^{pap}bp$	$N = 7$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 14\%$ $C_s = 0.4,\ C_r = 0.66$
g_{tu}^{pdw}	p_{ba}^{pba}	p_{ba}^{pa}	$N = 6$ $e_s = 16\%,\ e_p = 0\%,\ e_r = 16\%$ $C_s = 0.38,\ C_r = 0.64$
e^{2ln2}	a^{bbab}	a^{bbab}	$N = 5$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.45,\ C_r = 0.59$
$z^{\Sigma\ z}$	$a^{\Sigma\ b}$	$a^a b$	$N = 3$ $e_s = 33\%,\ e_p = 33\%,\ e_r = 0\%$ $C_s = 0.3,\ C_r = 0.99$
$\lambda n\ 2d_{Bkl}^3 sn\theta$	$bdbb_{bbb}^b aab$	$pdbb_{abb}^b aab$	$N = 11$ $e_s = 18\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.38,\ C_r = 0.87$
$a_{long}^{very} ME$	$a_{baap}^{aaap} bb$	$a_{baap}^{aaap} aa$	$N = 11$ $e_s = 18\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.39,\ C_r = 0.9$
$*\hbar\,\partial_{tu}\,\Phi_{xt}$	$bb_{ba}a_{ab}$	$pb_{ba}a_{ab}$	$N = 7$ $e_s = 28\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.39,\ C_r = 0.52$
$log_{2x}^{10} x$	$bap_{ba}^{bb} a$	$bapb^{b_a} a$	$N = 8$ $e_s = 0\%,\ e_p = 37\%,\ e_r = 37\%$ $C_s = 0.43,\ C_r = 0.69$
x_{max}	a_{aaa}	a_{aaa}	$N = 4$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.43,\ C_r = 0.95$

Actual Expression	Expected Result	Recognition Result	Statistics
$\sum_{ln2}^{lnX} e^{snp}$	$\sum_{bab}^{bab} a^{aap}$	$\sum_{bab}^{baa} a^a ap$	$N = 11$ $e_s = 9\%,\ e_p = 9\%,$ $e_r = 0\%$ $C_s = 0.41,\ C_r = 0.76$
$\sum_{ghm}^{p}$	$\sum_{pba}^{p}$	a_{p}_{p^{p a}}0	$N = 5$ $e_s = 40\%,\ e_p = 0\%,\ e_r = 40\%$ $C_s = 0.3,\ C_r = 0.61$
$*e^{\prod ln x}$	$a^{\Sigma\ baa}$	$a^{\Sigma\ baa}$	$N = 5$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.38,\ C_r = 0.81$
$\sum_{2xy}^{aN} log_{n\alpha} F$	$\sum_{bap}^{ab} bap_{aa}b$	$\sum_{bap}^{aa} bap_{aa}a$	$N = 12$ $e_s = 16\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.41,\ C_r = 0.79$
$*\bigcap_{a}^{Bn} \bigcup_{pP}^{} Y^m$	$\sum_{a}^{ba} \sum_{pb}^{} b^a$	\sum^{a a}_{p_{a^{a}}} _{p^{a}}	$N = 9$ $e_s = 44\%,\ e_p = 0\%,\ e_r = 22\%$ $C_s = 0.36,\ C_r = 0.71$
$x^{7u} \prod_{qF}^{Mv} E_{avg}$	$a^{ba} \sum_{pb}^{ba} b_{aap}$	$a^{ba} \sum_{p^a}^{aa} a_a ap$	$N = 12$ $e_s = 25\%,\ e_p = 8\%,\ e_r = 8\%$ $C_s = 0.41,\ C_r = 0.75$
$K_{\prod\ x}^{\prod\ y}$	$b_{\sum\ a}^{\sum\ p}$	$b_{\sum\ a}^{\sum\ p}$	$N = 5$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.36,\ C_r = 0.98$
$\sum_{\bigcup A}^{\bigcap d} mv^2$	$\sum_{\sum b}^{\sum b} aa^b$	$\sum_{p_a}^{p_p} aa^b$	$N = 8$ $e_s = 50\%,\ e_p = 0\%,\ e_r = 25\%$ $C_s = 0.34,\ C_r = 0.69$
$\varepsilon_{\alpha p}^{\Gamma\pi}$	a_{ap}^{ba}	a_{ap}^{ba}	$N = 5$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.4,\ C_r = 0.75$
$\varphi \bigvee_{Sv\ 0}^{4l} \bigwedge Tp$	$p \sum_{ba}^{bb} \sum_{b}^{} bp$	$\sum_{a_a}^{bb} \sum_{b}^{} \sum_{}^{} ap$	$N = 10$ $e_s = 30\%,\ e_p = 0\%,\ e_r = 20\%$ $C_s = 0.35,\ C_r = 0.81$

Table C.4: Results for testset3-1 (left) and testset3-2 (right)

Actual Expression	Expected Result	Recognition Result	Statistics
$a^{u^a}but_{a^n}$	$a^{a^a}bab_{a^a}$	$a^{a^a}bab_{a^a}$	$N = 8$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.47,\ C_r = 0.71$
$v^{s^A_u}r_{p^n}$	$a^{b}_{a^a}a_{p^a}$	$a^{b}_{a^a}a_{p^a}$	$N = 7$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.43,\ C_r = 0.99$
$ra^{p_{t^o}}r$	$aa^{p_b{}_a}a$	$\sum a^{pba}a$	$N = 6$ $e_s = 16\%,\ e_p = 16\%,\ e_r = 16\%$ $C_s = 0.38,\ C_r = 0.84$
$*\sum_{a^b}^{k_o} K$	$\sum_{a^b}^{b_a} b$	$\sum_{a^b}^{p_a} a$	$N = 6$ $e_s = 33\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.38,\ C_r = 0.98$
F_{x^2}	b_{a^b}	$b_a b$	$N = 3$ $e_s = 0\%,\ e_p = 33\%,\ e_r = 33\%$ $C_s = 0.38,\ C_r = 0.99$
S^{k_y}	b^{b_p}	$\sum^{p_p}$	$N = 3$ $e_s = 66\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.24,\ C_r = 0.99$
$e_{u^7}^{n_2}$	$a_{a^b}^{a_b}$	$a_{a^b}^{a_b}$	$N = 5$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.45,\ C_r = 0.99$
$n^{e^{s^{t_e d}}}$	$a^{a^{b_{ab}}}$	$a^{a^{b_{ap}}}$	$N = 6$ $e_s = 16\%,\ e_p = 16\%,\ e_r = 16\%$ $C_s = 0.38,\ C_r = 0.95$
$*3x_{s_0}\ 2y_{t_n}$	$ba_{a_b}bpb_a$	$ba_{a_b}bpb_a$	$N = 8$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 12\%$ $C_s = 0.45,\ C_r = 0.73$
$\prod_{u^l}^{v\ l} \sum_{\sum_a^b g}^{} A_r$	$\sum_{a\ p}^{a\ b} b_a$	$\sum_{a^p}^{b_a} a_a$	$N = 11$ $e_s = 27\%,\ e_p = 18\%,\ e_r = 0\%$ $C_s = 0.34,\ C_r = 0.99$

Actual Expression	Expected Result	Recognition Result	Statistics
$d^{a_{23}}vb_n$	$b^{a^b_{bb}}ab_a$	$b^{a^b_{bb}}ab_a$	$N = 8$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.46,\ C_r = 0.85$
$e^{\Pi^a_b{}_{2nx}}$	$a^{\sum^a_b{}_{baa}}$	$a^{b^a_{bbaa}}$	$N = 7$ $e_s = 14\%,\ e_p = 14\%,\ e_r = 0\%$ $C_s = 0.4,\ C_r = 0.82$
$D_{e_{e_P}}T^{h^{o^{u^g}_h}_T}$	$b_{aa_b}b^{a^p_b}$	$b_{a_a}a^{p^b_p}ap$	$N = 11$ $e_s = 54\%,\ e_p = 27\%,\ e_r = 27\%$ $C_s = 0.33,\ C_r = 0.76$
α^{Γ_π}	a^{b_a}	a^{b_a}	$N = 3$ $e_s = 0\%,\ e_p = 0\%,$ $e_r = 0\%$ $C_s = 0.4,\ C_r = 0.99$
$\sum_{\Sigma_1}^{\Sigma_2} \Pi_{x^2}$	$\sum_{b_a}^{b_b} b_{a^b}$	$\sum_{a_b}^{a_b} a_a b$	$N = 8$ $e_s = 37\%,\ e_p = 12\%,\ e_r = 12\%$ $C_s = 0.35,\ C_r = 0.97$
$\bigotimes_{\varphi^a}^{\psi^b} z_x$	$\sum_{p^a}^{p^b} a_a$	$\sum_{a^a}^{p^b} a_a$	$N = 7$ $e_s = 14\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.39,\ C_r = 0.98$
m_{g^h}	a_{p^b}	a_{pp}	$N = 3$ $e_s = 33\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.33,\ C_r = 0.95$
$b_{c^{n^0}}$	$b_{a^{a^b}}$	$b_{a^a}b$	$N = 4$ $e_s = 0\%,\ e_p = 25\%,\ e_r = 25\%$ $C_s = 0.42,\ C_r = 0.99$
$\bigcup_{a^Y\ r_l}^{m\ N} \bigcap \Delta$	$\sum_{a^b\ a_b}^{a\ b} \sum b$	$\sum_{a^a\ a_b}^{a\ a} \sum a$	$N = 9$ $e_s = 33\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.38,\ C_r = 0.95$
$\varphi_n \bigvee_{p_x}^{q_y} X\ Y$	$p_a \sum_{p_a}^{p_p} bb$	$\sum a \sum_{\sum^a}^{p_p} aa$	$N = 9$ $e_s = 44\%,\ e_p = 11\%,\ e_r = 22\%$ $C_s = 0.34,\ C_r = 0.92$

Table C.5: Results for testsetNL (left) and testsetHW (right)

Actual Expression	Expected Result	Recognition Result	Statistics
$d\Delta \quad HdxGdy$	$bbbbabbp$	$\sum aa \sum ab \sum$	$N = 8$ $e_s = 62\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.31,\ C_r = 0.84$
$\sum_{x}^{Y} e^{ln\,p}$	$\sum_{a}^{b} a^{bap}$	$\sum_{a}^{a} abaa$	$N = 7$ $e_s = 28\%,\ e_p = 0\%,\ e_r = 14\%$ $C_s = 0.37,\ C_r = 0.58$
$\bigcap_{*\ 3e}^{p^b} sp$	$\sum_{ba}^{p^b} ap$	$\sum_{ba}^{a^b} p_{\sum}$	$N = 7$ $e_s = 42\%,\ e_p = 0\%,\ e_r = 14\%$ $C_s = 0.37,\ C_r = 0.8$
$p^{g_b}\,h_{v,r}$	$p^{p_b} b_{a^a}$	$a^{ppp} a^a$	$N = 6$ $e_s = 50\%,\ e_p = 33\%,\ e_r = 33\%$ $C_s = 0.31,\ C_r = 0.99$
$N_k^{\sum ds}$	$b_b^{\sum ba}$	a_p^{paa}	$N = 5$ $e_s = 80\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.26,\ C_r = 0.66$
$e^{t\theta}$	a^{bb}	a^{bp}	$N = 3$ $e_s = 33\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.33,\ C_r = 0.99$
$\bigcup_{l \in I} A_l$	$\sum_{bab} b_b$	b_{bab}^{av}	$N = 6$ $e_s = 33\%,\ e_p = 0\%,\ e_r = 16\%$ $C_s = 0.34,\ C_r = 0.79$
$\sum_{b}^{q} x^n$	$\sum_{b}^{p} a^a$	$\sum_{p}^{p} a^a$	$N = 5$ $e_s = 20\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.39,\ C_r = 0.94$
$e^{\log_r k^n}$	$a^{bap_a b^a}$	a^{bapab^a}	$N = 7$ $e_s = 0\%,\ e_p = 14\%,\ e_r = 14\%$ $C_s = 0.41,\ C_r = 0.88$
$\cos 3\tau$	$aaaba$	$papba$	$N = 5$ $e_s = 40\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.35,\ C_r = 0.74$

Actual Expression	Expected Result	Recognition Result	Statistics
$\alpha \cos 3\pi$	$aaaaba$	$a \sum \sum bb_a$	$N = 6$ $e_s = 50\%,\ e_p = 0\%,\ e_r = 16\%$ $C_s = 0.31,\ C_r = 0.82$
$b_a p^{\tilde{a}}$	$b_a p^a$	$b_a p^a$	$N = 4$ $e_s = 0\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.44,\ C_r = 0.83$
$\sum_{0}^{N} b$	$\sum_{b}^{b} b^a$	$\sum_{p}^{p} b^a$	$N = 5$ $e_s = 40\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.34,\ C_r = 0.9$
$e^{x \ln 2}$	a^{abab}	$a^{ab_p b}$	$N = 5$ $e_s = 20\%,\ e_p = 20\%,\ e_r = 20\%$ $C_s = 0.4,\ C_r = 0.72$
$A \bigcap_{*} a^{x^3}$	$\sum_{b_p}^{b} a^{b^a}$	$\sum_{b_b}^{p} a^{b^a}$	$N = 7$ $e_s = 28\%,\ e_p = 0\%,\ e_r = 0\%$ $C_s = 0.39,\ C_r = 0.96$