Structured Editing of Handwritten Mathematics

Alexandra Sofia Ferreira Mendes

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy

School of Computer Science
May 2012

Abstract

Teaching effectively requires a clear presentation of the material being taught and interaction with the students. Studies have shown that Tablet PCs provide a good technological support for teaching. The aim of the work presented in this thesis is to design a structure editor of handwritten mathematics that explores the facilities provided by Tablet PCs. The editor is made available in the form of a class library that can be used to extend existing tools.

The central feature of the library is the definition of structure for handwritten mathematical expressions which allows syntactic manipulation of expressions. This makes it possible to accurately select, copy and apply algebraic rules, while avoiding the introduction of errors. To facilitate structured manipulation, gestures are used to apply manipulation rules and animations that demonstrate the use of these rules are introduced. Also, some experimental features that can improve the user's experience and the usability of the library are presented. Furthermore, it is described how to integrate the library into existing tools. In particular, *Classroom Presenter*, a system developed to create interactive presentations using a Tablet PC, is extended and used to demonstrate how the library's features can be used in some teaching scenarios.

Although there are limitations in the current system, tests performed with teachers and students indicate that it can help to improve the experience of teaching and learning mathematics, particularly calculational mathematics.

Acknowledgements

I would like to thank my supervisor, Professor Roland Backhouse, for all the support and feedback he gave me throughout my PhD. Thank you for always being available and for inviting us over to your house to spend wonderful afternoons chatting and enjoying the garden. I will never forget that. Thanks also to Hilary Backhouse for being so friendly and for always receiving us so well.

I would also like to thank Dr. José Nuno Oliveira for accepting to be my co-supervisor, for his support, and for his enthusiasm in the lectures and in the discussion of problems.

My sincere gratitude also goes to Dr. Luís Soares Barbosa for everything he has done for me. Thanks to him I have got the opportunity to start a PhD and his constant support and availability have been very valuable.

My thanks also goes to my examiners, Dr. Colin Higgins and Dr. Richard Verhoeven, for their feedback on my work.

I would like to express my gratitude to *Maplesoft* for giving me permission to use one of their recognisers.

A very special thanks goes to my GP, Doctor Simon Royal, for being the only doctor able to help me with the health problems that crossed my way during my PhD studies. He is the best doctor I have ever met.

I would also like to acknowledge the help of Nocas and Professora Marta for all the advice they gave me and for their friendship.

A very special thanks goes to my parents, Francisco e Rosa Mendes, who worked very hard all their lives to give to their children the opportunities and care that they never had. Pai e Mãe, sei que vocês sempre trabalharam muito duro para me darem as oportunidades e os cuidados que vocês nunca tiveram, e eu dou muito valor a isso. Muito

obrigada por tudo!

Finally, I would like to thank the most important person in my life: my beloved husband, João Ferreira. Without his love and support (and feedback) I would have never finished this work. He is the most loving and caring person that I have ever met. Thank you João for always being by my side, for always listening, for being there for me whenever I need, and for being the perfect husband and friend. You make my life worthwhile and I love you more than I thought possible.

A Fundação para a Ciência e a Tecnologia apoiou o trabalho desenvolvido nesta tese com a bolsa de investigação `SFRH / BD / 29553 / 2006`, financiada por fundos nacionais do MCTES.

FCT
Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

Contents

1 Introduction **2**

1.1 Writing mathematics using a computer 4

1.2 Calculational method 5

1.3 Objectives 6

1.4 An example task 9

1.5 Contributions 12

1.6 Structure and organisation 14

2 An overview of pen/touch-based technologies and structure editing **15**

2.1 Structure editing 15

2.2 Pen- and touch-based technologies 17

2.3 Software for handwritten mathematics 20

2.4 Limitations and tendencies of current technologies 23

3 The functional design **25**

3.1 Library versus standalone editor 25

3.2 Supported notations 26

3.3 Typeset versus handwriting 27

3.4 Handwriting recognition 28

3.5 Text editing versus structure editing 28

3.5.1 Basic editing operations 29

3.6 Manipulation rules . . . 30
3.7 Animating manipulation rules . . . 31
3.8 Use of gestures . . . 32
3.8.1 Gestures to trigger actions . . . 32
3.9 System versus user in control . . . 34
3.10 Combined writing of mathematics and text . . . 35

4 The technical design 36

4.1 Architecture . . . 36
4.2 Extendability and reusability . . . 37
4.2.1 Reusability . . . 38
4.3 Handling of strokes . . . 39
4.4 Character recogniser . . . 42
4.4.1 Bayesian network models . . . 43
4.5 Parsing recognition results . . . 44
4.5.1 Data structures for mathematical expressions . . . 50
4.5.2 Creation of structured trees . . . 56
4.6 Spatial indexing of handwritten expressions . . . 60
4.7 Structure editing . . . 62
4.7.1 Selection and copy . . . 62
4.7.2 Manipulation rules . . . 66
4.8 Animation support . . . 69
4.9 Gestures support . . . 69
4.10 Markup and binary outputs for mathematical expressions . . . 74
4.11 Experimental features . . . 77
4.11.1 Handwritten templates . . . 77
4.11.2 Automatic space adjustment . . . 78

5 Integration into external tools 79

5.1 Introduction . . . 79

5.2 How to integrate the library . . . 80

5.2.1 Module *Recognisers* . . . 80

5.2.2 Module *Parsers* . . . 82

5.2.3 Module *Structure* . . . 84

5.2.4 Module *Gestures* . . . 86

5.3 An Example: Extending Classroom Presenter . . . 87

5.3.1 Technical details . . . 93

5.3.2 Conclusions . . . 99

6 Evaluation and testing **100**

6.1 Software evaluation . . . 100

6.2 Evaluation by Teachers . . . 101

6.2.1 Results . . . 102

6.3 Evaluation by Students . . . 108

6.3.1 Results . . . 109

6.4 Testing the line division algorithm . . . 119

6.5 Conclusions . . . 123

7 Discussion and conclusions **125**

7.1 Current system . . . 128

7.2 Future work . . . 128

7.2.1 Improve reliability and usability . . . 128

7.2.2 Improve the recognition of handwritten mathematics . . . 131

7.2.3 Incorporate feedback and further testing . . . 131

7.2.4 Improve the extended Classroom Presenter . . . 132

7.2.5 Summary . . . 132

Definitions **132**

References **134**

A Teachers' Guide **143**

B Teachers' Questionnaire **152**

C Students' Questionnaire **155**

D Using the current system to write a proof **158**

List of Figures

1.1 Euclid's algorithm written using the GCL. 9

3.1 Circle gesture 32

3.2 Using the semi-circle gesture to apply a rule 33

3.3 Using the drag-and-drop gesture to apply a rule 33

4.1 System architecture 37

4.2 Usage of the Strategy pattern 38

4.3 Handwritten character 'H' 39

4.4 Extending the bounding box of 'i' 41

4.5 Context menu with recognition results 43

4.6 Operator Editor 45

4.7 Brackets used for hints 49

4.8 Rose tree that represents the expression $a+b$ 52

4.9 How a *Renderer* object draws ink strokes to a viewport 54

4.10 Rose tree that represents the expression $a + b + c \times d \times e + f \times g$ 54

4.11 Swap operation 55

4.12 Double-tap on the last addition operator 63

4.13 Double-tap on the first addition operator 63

4.14 Double-tap on the second addition operator 63

4.15 Double-tap on the multiplication operator 64

4.16 Check gesture 65

4.17 Symmetry . . . 66
4.18 Animation for the *Symmetry* rule . . . 70
4.19 Using the *SemiCircleRight* gesture to distribute multiplication over addition 71
4.20 Result of applying distributivity . . . 72
4.21 Selection of the expression to be used in *Leibniz* . . . 72
4.22 Application of *Leibniz* using the *Check* gesture . . . 72
4.23 Result of applying *Leibniz* . . . 73
4.24 The *Scratchout* gesture . . . 73
4.25 Gestures' Editor . . . 74
4.26 Templates selector . . . 78
4.27 Summation . . . 78

5.1 Classroom Presenter in use . . . 88
5.2 Classroom Presenter in use . . . 89
5.3 Intermediate step of a derivation of Euclid's algorithm . . . 89
5.4 Derivation of Euclid's algorithm (Part I) . . . 90
5.5 Derivation of Euclid's algorithm (Part II) . . . 91
5.6 Derivation of Euclid's algorithm (Part III) . . . 91
5.7 Derivation of Euclid's algorithm (Part IV) . . . 92
5.8 Derivation of Euclid's algorithm (Part V) . . . 92

6.1 Level of difficulty in using the features (0 - Very easy; 4 - Very difficult) . 103
6.2 Statement: The features provided by this software can help with teaching mathematics . . . 104
6.3 Do you think that animations would be useful for teaching? . . . 105
6.4 Do you think that play/pause features for the animations would be useful? 105
6.5 Statement: Gestures are a good way to trigger actions . . . 106
6.6 Statement: I prefer to manipulate a formula through the structure than manually moving the ink around to obtain the desired result . . . 106

6.7 Statement: I prefer to use the handwritten representation of formulae than their representation in typeset . . . 107
6.8 Statement: I find it easier to handwrite mathematics than writing it using a keyboard . . . 108
6.9 Statement: Input using a pen is more precise than using the fingers . . . 108
6.10 Did the software help you understanding the manipulations involved? . 110
6.11 Statement: This software can help me doing calculations by myself . . . 110
6.12 Statement: This software can improve my experience in the classroom . 111
6.13 Statement: This software can help me with learning calculational mathematics. . . . 111
6.14 Statement: This software can help me with learning general mathematics 112
6.15 If this software became available during your attendance to the Program Calculus course, would you have used it? . . . 112
6.16 Would you use this software for anything else? . . . 113
6.17 Statement: Being able to send answers from my computer to the teacher's makes it easier to participate in the lectures . . . 113
6.18 Statement: Being able to send answers from my computer to the teacher's makes the lectures more interactive . . . 114
6.19 Do you think animations would be useful to understand better how algebraic rules are applied? . . . 114
6.20 Do you think that play/pause features for the animations would be useful?115
6.21 Statement: Gestures are a good way to trigger actions . . . 115
6.22 Statement: The gestures used during the presentation are adequate to the operations to which they are associated . . . 116
6.23 Relation between the students' opinion about the calculational method and the software (method,software) . . . 117
6.24 Statement: I find it easier to handwrite mathematics than writing it using a keyboard . . . 118
6.25 Input using a pen is more precise than using the fingers . . . 118
6.26 Sample 1: Line division (algorithm 1) . . . 119

6.27 Sample 1: Line division (Ink.Divider) 120
6.28 Sample 2: Line division (algorithm 1) 121
6.29 Sample 2: Line division (algorithm 1) – intersecting lines 121
6.30 Sample 3: Line division (Ink.Divider) 122
6.31 Sample 3: Line division (algorithm 1) 122
6.32 Sample 4: Line division (algorithm 1) 123

CHAPTER 1

Introduction

Teaching effectively requires a clear presentation of the material being taught and interaction with the students. The more the students feel involved, the better they will follow and understand the lectures. Teaching mathematics is no exception, and being a subject that many students find difficult, a clear and interactive presentation becomes even more important. A good way to capture the attention of the students (or of an audience in general) is to show in real-time how the results presented are used or obtained. For example, showing in real-time the steps of a calculation helps understanding how the final result was attained and how to use the mathematical rules that were applied.

For many decades, the blackboard was used in lectures as the main tool of instruction. The blackboard is useful to keep a record of what is being discussed and to present the material. Its flexibility is one of its great advantages as it allows to easily add new information when it is needed. However, it has some drawbacks:

- the material has to be written during the lecture — which can lead to errors and uses valuable time that could be spent explaining and discussing the material;
- at the end of the lecture, the blackboard is rubbed off — the students have to copy the material during the lecture making it more difficult to pay attention to the lecturer's explanations; also, if the lecturer wants to use the same material for another lecture, they have to rewrite it again.
- when no space is left, parts of the material have to be cleared from the blackboard, making it more difficult to relate material given later in the lecture with material that was already cleared.

The ideal would be to write in advance all the material that is going to be presented, and leave for the lecture time only the parts that should be accomplished by discussion and interaction with the audience. Also, it would be good to provide at the end of the lecture a copy of the material presented, so that the audience could print it and study it. A presentation technology that helps achieve some of these points is the overhead projector where one places transparencies containing the lecture's material. For many years this was a very popular way of lecturing. However, transparencies are not easily edited and the discussion during the lectures is usually recorded on the blackboard, whose drawbacks were presented before.

In the last decade, the use of computers connected to data projectors has become the most popular presentation technology. The presentations commonly consist of electronic slides which are prepared in advance and easily made available after, or even before, the lectures. This is a clear and, usually, organised way of making a presentation. Nevertheless, the electronic slides are generally fixed and work more like a guideline to the lecture than as a medium for discussion and interaction; it is difficult to adjust them during the presentation which does not encourage interactivity. In mathematics lectures there is a need to provide examples and details according to the students responses and also to freely interact with the students and adapt the slides. Thus, electronic slides are not the most appropriate to teach mathematics. But the lack of flexibility of the electronic slides is not the only problem: as the traditional input devices for computers are the mouse and the keyboard, it is hard and laborious to write electronic slides containing mathematics. In fact, it is hard to write any kind of document containing mathematical notation using an ordinary computer.

The goal of this project is to design a structure editor of handwritten mathematics. The editor is oriented for Tablet PCs and its aim is to help with teaching mathematics by providing a way to manipulate the structure of mathematical expressions [Men08]. The main mathematical method supported by the library is the *calculational method* which is detailed in section 1.2.

The technical development of the editor is divided in two main parts. First, a class library (called *MST library*[1]) that implements mechanisms for creating and manipulating the structure of handwritten mathematical expressions is provided. The library also implements a mechanism that allows structure manipulations using gestures and a mechanism that allows structure manipulations to be shown step-by-step. The details

[1]MST stands for Math∫pad Tablet.

about this class library are presented in chapters 3 and 4.

Second, to demonstrate that the *MST library* can be easily integrated into other tools, and to illustrate the concepts implemented, the library was integrated into the prize-winning tool *Classroom Presenter*. The details about this integration are shown in chapter 5.

1.1 Writing mathematics using a computer

As mentioned in the previous section, it is hard to write mathematics using a computer. The difficulty is mainly due to its usual input devices: the keyboard does not support all mathematical symbols and the use of the mouse to click in buttons to input the symbols is time-consuming. To overcome this problem, languages like LaTeX and MathML were created and, in fact, they are very useful and widely used. However, their use requires a huge mental effort from the user to transform mathematics into valid LaTeX or MathML code.

Applications like Math∫pad [Ver00, BV97], Scientific WorkPlace [sw10] , MathMagic [mm12] and MathType [mt12] were also created to simplify the writing of mathematics. But their input devices remain the keyboard and the mouse, which still present some difficulties to the end user.

Clearly, writing mathematics using pen and paper is much more natural[2] and easier than using a computer. But one problem with this approach is that the paper is not digital. It is possible to scan the paper (off-line data), to store it in a digital format, and to recognise the handwriting directly from the image. However, this is not dynamic and does not offer the possibility of editing the piece of paper as much as one would like or need.

It is of great interest to combine computer flexibility with handwritten simplicity to write and present mathematics, and that can be achieved by using a Tablet PC or any other device that provides ink collection (usually called pen-based technologies) — i.e. that allows input of handwriting. Although these devices facilitate the writing of mathematics, the available software does not allow us to edit and manipulate the structure of the mathematics used in areas such as algorithmic problem solving [Bac11, VB99, Bac03, Bac01]. Existing applications are more concerned with the recognition and

[2]In this thesis, the adjective natural is often used to describe features or methods that are simple to use and do not interfere with the user's thought.

evaluation of mathematical expressions than with the way of doing mathematics. Also, the majority of these tools do not allow combined writing of mathematics and text, which is needed to write mathematical presentations. In general, tools for handwritten mathematics do not recognise text while tools for handwritten text do not recognise mathematics.

This project is aimed at fulfilling the need of writing and presenting mathematics in an effective, flexible and also reliable way. The purpose is to provide a base for a system that assists in the process of doing mathematics and, in particular, provides structured manipulation of mathematical expressions. The system is oriented to the mathematics used in algorithmic problem solving and will be capable of recognising and editing mathematics written on the Tablet PC's screen. However, the focus of this work is not on recognition but, instead, it is on how to create structure from recognised handwritten mathematics and on how to allow its manipulation.

1.2 Calculational method

Two decades of research on *correct-by-construction* program design have created a new discipline of algorithmic problem solving and shed light on the underlying mathematical structures, modelling, and reasoning principles. Starting with the pioneering work of Dijkstra and Gries [DS90, GS93], and in particular, through the development of the so-called *algebra of programming* [BM97, BH93], a *calculational style* [Bac01, vG90] emerged, emphasising the use of systematic mathematical calculation in the design of algorithms. The realisation that such a style is equally applicable to logical arguments [DS90, GS93] and that it can greatly improve on traditional verbose proofs in natural language has led to a systematisation that can, in return, also improve exposition in the more classical branches of mathematics. In particular, lengthy and verbose proofs (full of *dot-dot* notation, case analyses, and natural language explanations for "obvious" steps) are replaced by easy-to-follow calculations presented in a uniform format[3]. Moreover, each step of a calculational proof is usually accompanied by a hint justifying the validity of that step. The following is an example of a calculational proof:

[3]The format used is due to Wim Feijen.

$$\begin{array}{lll} & A & \\ = & & \{\quad \text{p} \;\} \\ & B & \\ \Leftarrow & & \{\quad \text{q} \;\} \\ & C & \end{array}$$

the above is written to prove that $A \Leftarrow C$. A, B and C are expressions, and p and q are hints why $A = B$ and $B \Leftarrow C$, respectively. Some relevant advantages of this format are that the hints reduce the search space, there are no repeated intermediate expressions, and it can be immediately concluded that $A \Leftarrow C$, just by looking at the first and last expressions and at the relations connecting them. Another important aspect of this format is that it *forces* the writer to provide explanations for each step.

Furthermore, while many mathematicians transform formulae only after interpreting them, exponents of the calculational method prefer to work with uninterpreted formulae and manipulate them according to their symbols and associated rules (i.e., let the symbols do their work). One of the incentives for the work presented in this thesis is the belief that the systematisation of a calculational style of reasoning, proceeding in a formal, essentially syntactic way, can greatly improve on the way proofs are presented. This view is supported in a recent study that, through a quantitative analysis, shows that students seem to prefer or understand better calculational proofs [FM09].

1.3 Objectives

Some studies have shown that the use of the Tablet PC in lectures can improve teaching [Cro08]. A study made using *Classroom Presenter*, a system developed to create interactive presentations using a Tablet PC, indicates that the use of a Tablet PC allows the lecturer to cover more material in the same period of time or to spend more time explaining and discussing the displayed material. Also, the same study claims that the students think that this is a good teaching method. More details of this study are presented in [Oli05]. Similar studies, that also support the use of Tablet PCs in the classroom, are detailed in [Fit04] and [SAHS04].

Tablet PCs are now widely available with handwriting recognition built into the software. Generally, the text recognition software is extraordinarily effective but it falls down badly when mathematical expressions (including computer programs) are included in the text. Moreover, because the internal representation of what has been written is hidden, it becomes difficult to modify the handwriting. Even worse, modifications can be unreliable. In the last few years, much effort has been done to improve mathematics recognition and many projects on recognition of mathematical expressions are currently taking place. Some of these projects are mentioned and briefly detailed in section 2.3.

The goal of this project is to develop a system that will assist in teaching the dynamics of algorithmic problem solving [Bac11, VB99, Bac03, Bac01]. The system must allow straightforward, flexible and reliable manipulation of documents containing substantial amounts of non-standard mathematical expressions [GKP94, vG90, GS93, GFvGM90] (non-standard in the sense of not being the sort of mathematics that is a well-accepted part of pre-university mathematics education).

The Math∫pad system [BV97] developed by Richard Verhoeven under the direction of Roland Backhouse, and first released in the mid 1990s, is a structure editor for mathematical documents that has proved very convenient for writing research articles across a wide spectrum of computing science. But of course, given the time that it was developed, it assumes keyboard/mouse input, and keyboard/mouse manipulation of documents. The goal is to develop a system akin to the Math∫pad system that uses handwritten input and pen gestures to manipulate a document. However, this system will be oriented to the teaching and presentation of mathematics rather than the preparation of technical documents in a conference style format. The benefits for teaching algorithmic problem solving with this system can be enormous. It will be possible to demonstrate the dynamics of problem-solving in a manner that improves on blackboard-style teaching by exploiting the reliability of computer software in copying and manipulating structured information.

As previously mentioned, the system should provide flexibility and reliability in the manipulation of documents containing mathematical expressions and in the syntactic manipulation of formulae. The main object of the syntactic manipulation features will be algebraic calculations. For example, one of the goals is to apply actions to a particular expression automatically — automatically in the sense that the user does not have to determine the result of applying the action. As an example, consider the following expression:

$$a \vee (b \wedge c)$$

Knowing that $\vee$ distributes over $\wedge$, it would be useful to apply the distributivity rule automatically rather than manually. Not only would it avoid the introduction of errors, but it would also require less effort from the user. By using a button or a gesture, the distributivity rule would be applied and the following result obtained:

$$(a \vee b) \wedge (a \vee c)$$

The way this will be achieved is detailed in section 4.9.

The system developed does not know which actions can be applied to a certain expression. It only has the general knowledge of how to apply the rule. Checking the validity of applying it to the expression is the responsibility of the user. The user should be in control of the mathematics being used. As a result, the library allows the rules to be applied freely, allows the definition of new binary operators in runtime, and also allows the redefinition of binary operators (that is, changing the operators precedence and the way that they associate).

The system allows the application of several known rules (like distributivity, associativity and symmetry) automatically. This ensures some of the reliability and flexibility desired. Moreover, it helps to avoid the introduction of errors since the system (if everything goes as expected) will apply the rule exactly as instructed.

Another feature included in the system is a limited form of combined writing of text and mathematics that can be used to annotate algorithms and to justify steps in calculations.

Furthermore, it provides facilities to write and edit algorithms and calculational proofs (mentioned in section 1.2). As said before, this system will be used, in particular, to teach algorithmic problem solving and the interactive nature of the system is crucial to achieve this goal. In particular, it supports the writing of algorithms using the *Guarded Command Language (GCL)* [Dij76]. Figure 1.1 illustrates an example of an algorithm written in GCL.

In summary, this project has the following goals:

- To develop a structure editor — in the shape of a class library — for structural editing of mathematical expressions using handwritten input. The editor is oriented to the mathematics that is used in algorithmic problem solving [Zei99, Bac03, Pol57, GKP94], but is also general-purpose. The library is a proof of concept, capable of being further developed into a professional product;

```
$\{M > 0 \wedge N > 0\}$
$m, n := M, N;$
$\{m \ \mathbf{gcd}\ n = M \ \mathbf{gcd}\ N\}$
$do$
  $n < m \rightarrow m := m - n$
$\Box$
  $m < n \rightarrow n := n - m$
$od$
$\{m = n = m \ \mathbf{gcd}\ n\}$
```

Figure 1.1: Euclid's algorithm written using the GCL.

- Demonstrate the use of the library's features to support some teaching scenarios for algorithmic problem solving presented in [Fer10];
- Integrate the class library into an external tool to show that it can be easily used by other programmers and researchers.

1.4 An example task

As mentioned in the section 1.3, one of the goals of this project is to support the teaching of mathematics. Examples of what can be taught in a maths lecture, together with suggestions on how to teach it, are presented in the teaching scenarios shown in [Fer10]. To illustrate how the system can be used during a lecture, a demonstration of how it can be used to help with the presentation of *Scenario 1* in [Fer10] is shown. The problem presented in *Scenario 1* is the following:

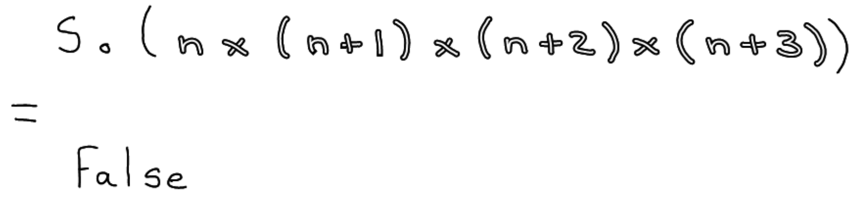
Problem: Prove that the product of four consecutive positive natural numbers cannot be the square of an integer number.

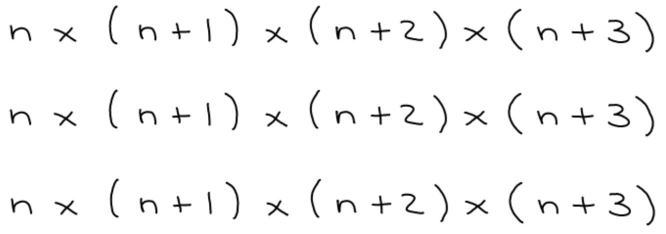
Following the solution presented in this scenario, the first thing that the teacher would write would be the goal:

$$S\circ(n\times(n+1)\times(n+2)\times(n+3))$$
$$=$$
$$\text{False}$$

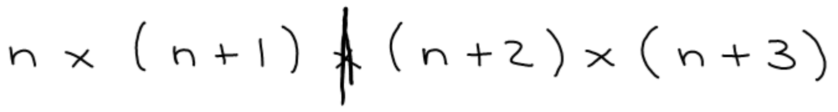
where $S.n$ is defined to be true only when n is the square of an integer. In order to provide a justification for why the predicate $S.(n\times(n+1)\times(n+2)\times(n+3))$ is false, the teacher should manipulate this predicate until it gets to a point where it is clear that the predicate is false. Having nothing else to do, the teacher should start by manipulating $n\times(n+1)\times(n+2)\times(n+3)$ by using the property that multiplication distributes over addition. As distributivity can be applied to this expression in three different ways, it is important that the right choice is made. In the scenario, it is recommended that the teacher asks the students which alternative should be chosen. To decide what is the best alternative, it would be desirable to copy the expression three times and apply distributivity to each copy separately. To do that using the system, the teacher can follow the following steps:

1. continuously perform a double tap over the multiplication operator to select the expression inside brackets (as distributivity will be applied to this subexpression); press and release the side button of the pen to make a copy of the selection (alternatively, the *Check* gesture can be used); make a tap in each position where the copies should be placed (this will put a copy of the selection starting at the point of the screen that was tapped):

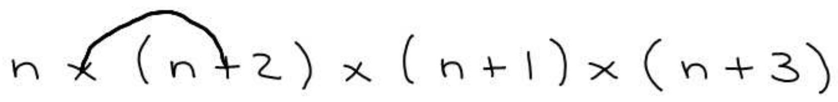



n × (n+1) × (n+2) × (n+3)
n × (n+1) × (n+2) × (n+3)
n × (n+1) × (n+2) × (n+3)

2. use symmetry of multiplication to put in place the expression that will be distributed (e.g., in order to distribute $n\times$ over $(n+2)$ in the expression $n\times(n+1)\times(n+2)$, the symmetry of multiplication is used to move $(n+2)$ to the immediate right of $n\times$ obtaining $n\times(n+2)\times(n+1)$) — that is done by using the *UpDown* gesture over the operator between the two expressions to be swapped; note how the use of symmetry, which is usually left implicit, is made explicit by the tool:



3. use the *SemiCircleRight* gesture to apply distributivity — the semi-circle should start over the operator that is to be distributed and finish over the operator through which it will be distributed (alternatively, the teacher can select the operator to be distributed, drag it, and drop it over the operator through which it will be distributed):



The result obtained is:

$$((n \times n) + (n \times 2)) \times (n + 1) \times (n + 3)$$

4. for each copy, apply symmetry and distributivity several times and in different ways. The use of strict syntactic manipulation results in the following expressions:

$$((n \times n) + (n \times 1)) \times ((((n \times n) + (2 \times n))) + (((n \times 3) + (2 \times 3))))$$
$$((n \times n) + (n \times 2)) \times ((((n \times n) + (n \times 1))) + (((3 \times n) + (3 \times 1))))$$
$$((n \times n) + (n \times 3)) \times ((((n \times n) + (1 \times n))) + (((n \times 2) + (1 \times 2))))$$

Note that, as this system does syntactic manipulation, expressions like $n \times n$ and $n \times 1$ are not simplified. However, the user can use *Leibniz* to simplify them.

Now, having the results of applying distributivity in three different ways, the teacher can ask the students which option they think is the best to use in this problem. The discussion should lead them to choose the one that is more symmetrical[4], that is,

$$(n^2 + 3 \times n) \times (n^2 + 3 \times n + 2).$$

This example shows how some of the features of the library can help with teaching. The aim of this example is to give the reader a better idea of the type of tasks that this library supports.

1.5 Contributions

The main contributions of this thesis are the functional and technical designs of a structure editor for handwritten mathematics with an emphasis on algebraic calculations

[4]In general, algebraic symmetry is a technique useful to simplify calculations. For more details, the reader is referred to section 3.2 of [Fer10].

and oriented for teaching presentations. The central feature of the editor is the definition of structure for handwritten mathematical expressions. This allows syntactic manipulation of the expressions making it possible to accurately select, copy and apply algebraic rules to expressions while avoiding the introduction of errors. To facilitate the structure manipulation, gestures are used to apply manipulation rules. In general, the gestures for each rule match the visual idea that is usually associated with the application of the rule (for instance, for distributivity a semi-circle starting at the operator that will be distributed and ending over the operator through which it will be distributed is used — this matches the visual idea that one operator will be propagated over the other). This makes the use of gestures more natural than the use of buttons to trigger the rules. Furthermore, it is possible to edit the association between gestures and rules so that users can customise the system to match their own "visual idea" of the rule. In the context of teaching, applying an algebraic rule and obtaining the result instantaneously may not be good enough. With this in mind, a mechanism to support the animation of certain structure manipulations is provided. The purpose of these animations is to show, step-by-step, how a rule is applied to an expression. The feedback from students suggests that animations can help students understand how the rule works (see chapter 6).

The main end-product is a class library that implements the features described above. This class library is implemented in C# and was created in a way that makes it easy to be used and extended by other programmers. In particular, it can evolve with new technological developments: for instance, it is possible to connect other parsers and recognisers to it; this is advantageous since the area of handwritten recognition is still evolving.

To illustrate how these concepts can be used in existing applications, the class library is integrated into *Classroom Presenter*, a Tablet PC based system for ink based teaching presentations and classroom interaction. Also, some experimental features that may be useful for teaching and writing presentations, like user-defined handwritten templates and automatic space adjustment inside expressions, were developed (details will be presented in Chapter 4).

1.6 Structure and organisation

Before presenting the technical and functional designs of a structure editor for handwritten mathematics, chapter 2 explains in more detail what is meant by structure editing and provides an overview of pen- and touch-based technologies currently available. A list of tools that support recognition of handwritten mathematics is shown and the limitations and tendencies of current technologies are discussed.

In chapter 3, the functional design of the system developed is provided. The requirements of the project and the decisions that were made are described.

Chapter 4 describes the technical design of the system. The most important features are described and details of their implementation are given. The chapter ends with the description of experimental features that may be further developed in the future.

Because the final product of this project is a class library, in chapter 5 it is shown how external tools can import and use the features provided by the library. To illustrate that the library can be easily integrated into other tools, the chapter also describes its integration into the prize-winning tool *Classroom Presenter*.

Chapter 6 presents the results obtained from usability and suitability tests that were performed to assess if the library's features are usable and suitable for their intended purpose. The testing was carried out with both teachers and students.

Finally, in chapter 7, a discussion of the achievements and the limitations of the work developed is shown. After demonstrating how the current prototype supports the initial goals, some directions for future work are presented.

After the conclusion, some definitions of important terms that are assumed to be known (and are not defined within the body of the thesis) are presented.

CHAPTER 2

An overview of pen/touch-based technologies and structure editing

This chapter explains what structure editing is and presents some examples of editors that provide some sort of structure editing capabilities. An overview of the pen- and touch-based technologies currently available and a list of tools that support recognition of handwritten mathematics are provided. Limitations and tendencies of current technologies are briefly discussed.

2.1 Structure editing

The purpose of an editor is to allow interactive creation and modification of documents. Although most editors solely have the knowledge of how to display the documents that they support, some are aware of the document's underlying structure — these are called *structure editors*. In a structure editor, the content of its documents is stored in a structured way so that actions can be performed against that structure.

Nowadays, people often deal with structure editors without even noticing. For example, in text editors that support spell-checking, when someone writes a word that the system does not recognise, it highlights that word and suggests other terms; if someone starts a sentence with a small letter, the system tries to capitalise it, as it thinks that this is the structure that the document should have. Thus many text editors are structure editors. Spreadsheet applications are another example. A spreadsheet is made of multiple cells that together form a grid of rows and columns. Being aware of this structure makes it possible to select columns of numbers and calculate their average, and rows can be sorted according to some criterion.

Other examples of structure editors are tools like *Vim* [vim12] and the integrated development environment (IDE) *Microsoft Visual Studio*. These tools support, for instance, syntax highlighting and bracket matching, which help when writing in a structured language — such as a programming language — by making structures and syntax errors visually explicit. These features are made possible because the tools are aware of the underlying structure of the documents they support.

The tools mentioned above are not "advertised" as structure editors. In fact, it is unusual to find tools that clearly state that they are structure editors, since the structure editing happens "behind the scenes". One tool that is clearly "advertised" as a structure editor, is the Math∫pad system [BV97]. Math∫pad is a structure editor for mathematical documents that has proved very convenient for writing research articles across a wide spectrum of computing science. The goal of this system is to make it possible to write mathematical documents in which mathematics and text are completely integrated. It aims to assist in the process of doing mathematics rather then obtaining results. Its main features are expression manipulation and a system for defining and using structured templates (called stencils).

In terms of expression manipulation, Math∫pad provides functions for rearranging expressions, and for finding and replacing mathematical expressions with a given structure. The functions provided for expression manipulation are *Copy*, *Swap*, *Reverse*, *Distribute*, *Factorise*, *Group*, *Ungroup*, *Apply*, *Find* and *Rename*. These are applied by using multiple selections and by choosing from a menu the function to be applied.

Regarding the Math∫pad stencils, they are files describing the visual and logical structure of notational elements in a document. This feature is very important because it does not restrict the user to a certain set of notations. The user can define and redefine their own notations. This is important in research since new theories may involve new notations. Also, as it is impossible to provide a system that supports every single existent notational convention, the fact that anyone can define their own notations makes this system widely usable and flexible.

The Math∫pad system has inspired the project presented in this thesis. The ultimate goal of Math∫pad was to provide a system that could eliminate the use of pen and paper for the calculations of the intended users of the system. However, that did not happen and they believe the main problem lies with computers and not with Math∫pad (see page 15 of [BV97]). One of the main problems with the "common" computer is that it receives its input through the keyboard and mouse. The keyboard is very limited in keys and it does not support all the mathematical symbols. To provide support for all the symbols, one would have to create many combinations of keys or different writing modes, which can be very confusing. Using the mouse to select symbols from a menu can be a solution, but it is very time-consuming. Also, both solutions interfere with the users thought as the users have to focus on the input of mathematical symbols rather than on the calculations. In an attempt to address these problems, this project was born. The system developed aims at making the input of mathematics easier by providing handwritten input. Its principal goal is to make it more intuitive to apply rules in a calculation and to show to an audience how the rules are applied. In the rest of this thesis, it is shown how these goals are met. However, as the reader will have the opportunity to see, the system is still some steps away from providing simple input for mathematics.

2.2 Pen- and touch-based technologies

In the last few years, the interest for hardware tools that allow alternative input to the keyboard and mouse has grown quite spectacularly. The release of touchscreen gadgets into the market, specially smartphones, has made this type of technology very popular. These gadgets have had a great reception as their interface is easy to use and most people already know how to use them. Moreover, with the recent release of touch tablets into the market (the iPad, Samsung Galaxy Tab, etc), it seems that the industry is moving towards pen- and touch-based technologies. However, technology that allows alternative input has been around for quite a long time: the first Tablet PC appeared in the 1950s!

Throughout this thesis, when the term *Tablet PC* is used, it refers to a computer with an *Electromagnetic Digitizer*. With this type of digitizer, a special pen is needed as a pointing device. This pen is a stand-alone device containing all the circuitry needed to interact with the computer. Other types of digitizers, which will be referred as *touch-screens*, accept as pointing device any object (finger, normal pen, etc). These digitizers have some disadvantages. For instance, they only detect actions when the pointing device touches the screen — no hovering is detected. Also, as no special device is used, the input is too imprecise — specially when it concerns to handwriting.

Many tools that allow input from stylus, finger or some other device to touch the screen, only allow one input touch at a time. However, some tools allow two or more input touches simultaneously — these are called multi-touch devices[1]. The multi-touch ability is mostly used in visual manipulation of contents (resizing of images, rotation of objects, navigation through maps, etc) as it provides a very intuitive interface for these tasks.

Nowadays, some computers are equipped with both a digitizer that requires a stand-alone pen and a touchscreen. A problem with these machines is that, when using the pen, the user tends to rest his wrist against the screen which will trigger actions that the user does not want to start. Thus, the touchscreen tends to interfere with the use of the pen. This situation may change soon as Microsoft Research is currently trying to combine the use of pen and finger to increase functionality and improve user experience. The name of the project that explores this is *Manual Deskterity* [HYP$^+$10] and it investigates how one should use pen and touch in interface design. The conclusion they have reached is the following: the pen writes, touch manipulates, and the combination of pen and touch yields new tools. (Their system already overcomes the problem of "wrist resting against the screen".) The techniques explored in *Manual Deskterity* were implemented in the *Microsoft Surface* [Mic12] with a special pen constructed for the purpose.

The *Microsoft Surface* is a multi-touch computer that responds to natural hand gestures and real-world objects. The key features of this system are:

- users can use their hands to grab digital information and use touch and gestures to manipulate the on-screen content;
- it supports multiple users — several people can use the system simultaneously and interact with each other;
- it supports multi-touch;
- and finally, it supports object recognition — users can place real-world objects on the screen and the system will recognise them.

The *Microsoft Surface* is already in use in many companies. This device, together with the *Manual Deskterity*, are possibly a preview of what computer systems will look like

[1]Some authors define multi-touch as the ability to simultaneously register three or more distinct input touches. However, in this thesis, the term multi-touch is used to refer to the ability to register more than one input touch at a time

in a near future.

Another example of a system that combines pen and touch is the *Touch & Write* [LREND10]. The *Touch & Write* is a novel rear-projection tabletop that combines infrared technology for the normal touching with the digital pen technology for high resolution handwriting. This allows implicit detection of pen and touch use, which is good to create systems where the touch and pen do not interfere with each other. Similarly to the *Manual Deskterity* approach, they use in their software (*LeCoOnt*) touching actions for visual manipulation and pen-actions for drawing, for connecting concepts, and for handwriting. The *LeCoOnt* is concept mapping software.

The touch- and pen-based technologies are used in many tasks. They are used, for example, for geometric constructions [MK04], for styling design of 3D geometry [KDS06], sketching in the hardware-software integrated interactive product design process [Nam05], for creating dynamic mathematical illustrations [LZ04], for 3D role modelling for children [LWD08], and to assist web designers in the early stages of web design [LNH$^+$01]. An area where these technologies have attracted special attention is education. In particular, the Tablet PC has attracted much attention as an aid for teaching, as it can be used with a projector to allow interactive presentations — which has been shown to be successful with the students (this has been mentioned in more detail in section 1.3). This success possibly led to the creation of *interactive whiteboards*.

An *interactive whiteboard* (IWB) is a large display that is connected to a computer and a projector. The screen of the computer is projected onto the display and the user can control the computer by using a pen, finger or other device against the whiteboard. This resembles the traditional blackboard while bringing to the classroom/workspace the computer capabilities. Other than in the classroom, the IWBs are used in a variety of settings. These include corporate board rooms, brainstorming sessions, collaborative live training and more. IWBs are very useful to promote interactivity with an audience.

In conclusion, there is a wide variety of technologies for pen and touch input. These technologies are used in many distinct areas and they are getting better and better. These technologies are now rapidly evolving and it is quite difficult to keep up with all the developments in the area. The goal of this section is to give the reader an idea of what is currently available.

2.3 Software for handwritten mathematics

In this section, some of the tools and packages that are capable of receiving handwritten mathematics as input and recognise it, are presented. For each of these tools, a brief summary is given.

MathJournal [mju12, WD03] is an interactive tool for the Tablet PC that provides a natural and intuitive environment for mathematical and engineering problem solving. The software recognises, interprets and provides solutions for handwritten or hand-drawn mathematical and engineering constructions. This tool recognises a great number of mathematical symbols. It evaluates several kinds of mathematical expressions and, given an expression, plots the corresponding graph. This application is available for the Windows operating system.

JMathNotes [jmn11] is a JAVA application that recognises mathematical expressions written on the screen and provides manipulation and correction of the expressions via menus and gestures. It produces LaTeX and generates output suitable to use in word processors and symbolic computation. It also provides algebraic manipulation through the computer algebra system Mathematica. This application is free software and can be modified or/and redistributed under the terms of the GNU General Public License. It can be used in Windows, Linux or any other Unix platform. This project seems to have been abandoned since its webpage is no longer available and no current information about this project could be found.

Natural Log [nlu03, Mat99] is an applet written in JAVA that was created as a demonstration tool for the concepts presented in Nicholas Matsakis' Master Thesis. It recognises all the basic mathematical symbols, some Greek letters, summations, fractions and square roots. It does not recognise integrals nor subscript notation. This applet produces LaTeX and MathML and does not evaluate mathematical expressions.

The software package **Caltech Interface Tools** (CIT) [cit12] is free software and recognises handwritten mathematical characters. The CIT code requires training data which creates user specific handwriting profiles. Little information can be found about this software package and it seems to be an abandoned project. However, in 2004, the creators of the **Math Speak & Write** tool have created a DLL (Dynamic-link library) that contains all of the CIT handwriting recognition code and makes it easily accessible through a small number of functions.

Math Speak & Write [msw12] system is designed to facilitate mathematics input using both voice recognition and handwriting recognition. This system was written for Microsoft Windows in the C# programming language. It uses a combination of Microsoft's Ink system to collect ink strokes and their DLL version of the CIT Handwriting Recogniser for the recognition of the handwriting. For the speech recognition they use the Microsoft's Speech SDK 5.13 with a custom grammar.

Freehand Formula Entry System [ffe05, Smi99] allows the freehand entry and editing of formulae using a pen and tablet. Automatic handwritten formula recognition is then used to generate a LaTeX command string for the formula. It is implemented in C++ and Tcl/Tk and it is distributed under the GNU General Public License. This system recognises input using a character recogniser (the CIT, mentioned above) and an expression parser (DRACULAE). It can be used either in Linux or Windows (through Cygwin [cyg12]). The latest version of this system was released in May 2004.

Java Interactive Mathematical Handwriting Recogniser (JIMHR) [jim12] is a tool that processes handwritten input through a mouse or a pen and outputs the corresponding mathematical formula in real-time as an image and either as LaTeX or MathML. This program allows the user to correct any mistakes resulting from misclassification or to rearrange the symbols to suit their needs. JIMHR runs in Unix, Linux and Windows. This system is based on the `Natural Log` and `Freehand Formula Entry System` that were mentioned previously.

The **MathBrush** [mbu12, LLM$^+$08] system allows the user to enter mathematical expressions with a pen, recognise them and send the expressions to a computer algebra system which will provide a final result. This system was designed to provide an environment for experimenting with the various components needed for doing mathematics with pen-based devices. This tool has some gestures defined and they were chosen to be similar to the ones that the user uses while writing using pen and paper and also to be consistent with other pen-based applications. An important feature of this tool is that it provides logging of mathematical manipulations. This means that it keeps track of all the user's actions in one math sheet. This is a currently active project and its recognition capabilities seem very accurate.

Another existing tool is **Infty Editor** [ieu12, FKS03]. This editor has online recognition of handwritten mathematical expressions and as soon as a character is written, it is automatically rewritten as neat strokes in an appropriate position and size. This editor provides output in several different formats: LaTeX, MathML, PDF, etc. This system can communicate with the computer algebra system `Mathematica` and it is planned that in the future it will communicate with various computer algebra systems.

MathPad^2 [mat06, LZ04] is a system for the creation and exploration of mathematical sketches. It is based on a novel concept of mathematical sketching, making dynamic illustrations by combining handwritten mathematics and free-form diagrams. MathPad^2 is designed so a user can create simple illustrations as if they were working with pencil and paper.

MathPaper [mat12, ZMLL08] is another tool that recognises handwritten mathematics. It allows free-form handwritten entry of multiple mathematical expressions to provide symbolic and numerical computational assistance. It uses real-time recognition and supports gestural and widget-based interactive editing. This tool also explores novel approaches to entering and manipulating matrices and allows handwritten entry of mathematical algorithms.

AlgoSketch [LMZL08] was created in the context of the MathPaper project and it is a pen-based algorithm sketching prototype with supporting interactive computation. It allows writing and editing of 2D handwritten mathematical expressions in the form of pseudocode-like descriptions to support the algorithm design and development process.

Starpad SDK [sta12] provides a convenient interface for a broad layer of pen-centric functionality in addition to some research functionality. It includes an interface to stroke-level operations, a recognition library for handwritten math and gestures, some UI techniques such as GestureBar [BZW$^+$09], and a pen- and gesture-based application shell supporting selection, undo, zooming, text input, images, and save/load. This project can be used freely for non-commercial use.

MoboMath [mob12] allows handwritten input of mathematics and translates it into a formatted layout that can be copied and used in Microsoft Word, Maple, MathML, TEXand LATEX, Images, and more. It includes a set of built-in editing tools, that allow the user to add, delete, correct, or rearrange any part of an expression using the pen. It makes use of gestures to perform some actions (for example, the recognition is triggered by a *right-down* gesture).

Detexify [det12] is a free-software symbol recogniser written in Ruby. It recognises individual symbols and it translates them directly to the correspondent LATEX code. Although this is a very recent project, it already supports many symbols that are used in algorithmic problem solving (for example, it is one of the few recognisers that recognises correctly the '$\langle$' symbol used in quantifiers). This means that if the project becomes more mature, it may be adapted to work with the tool described in this report.

Maple [map12] is a very powerful mathematical computation engine that allows the use of handwriting to sketch symbols in order to help finding easily the symbol that the user wants to use. The symbol recogniser searches through all of the available symbols and offers a choice of probable matches. This symbol recogniser is currently being used in the project described in this report.

Finally, the **Mathink**[SW06] system provides a user interface allowing digital ink to be collected from a pen, mouse or other device. It provides immediate recognition results for the handwritten expressions (after an adjustable time delay) along with other high ranked candidates (so that the user can easily select other alternative). The recognition results accepted by the user are displayed on top of or instead of the original ink. After the user gets the correct result, the typeset expression is parsed to a standard mathematical format, such as Presentation MathML. At this point the user can choose to send it to a symbolic computation back-end to perform further computation such as evaluation, simplification and solving. In addition to mathematical expressions analysis, the Mathink system offers a drawing mode (where ink is recognised as geometrical shapes) and also a "free ink" mode (where ink is not sent to the recogniser). This system also allows addition of new symbols to its database.

The main focus of most of the tools presented is mathematical recognition (or symbol recognition). Some, like **MathJournal**, support the evaluation of mathematical expressions. However, none of these tools provides facilities for *structured* manipulation of mathematical expressions and none of these support the mathematics used in algorithmic problem solving.

2.4 Limitations and tendencies of current technologies

Although pen- and touch-based technologies have evolved a lot in the last few years, these technologies still have some limitations.

Concerning handwriting recognition, current recognition systems do a good job but, as handwritten styles differ from user to user, it is basically impossible to provide systems that are fault free. Moreover, as there are many symbols and objects to be recognised, it is only natural that some may be mis-interpreted as other symbols/drawings that are similar in representation. However, current interfaces already provide good ways to let the user be in control of what is recognised by letting them choose alternative recognition results.

Regarding the use of gestures (through the use of pen or touch), there is a limit to the number that should be used. Some systems use too many gestures, making it almost impossible for the user to use them all. In general, users cannot remember too many gestures easily. And this is not the only problem: gestures may interfere with each other and with the user's tasks. Having too many gestures defined will cause accidental touches on the screen and attempts to handwrite, to be mis-interpreted as gestures triggering unwanted actions. This can easily become a serious usability problem. This opinion is supported by the decision of Microsoft to provide just 9 touch gestures in Windows 7.

Current technologies are showing a tendency to support touch and discard the use of pen. However, pen-based systems provide a more precise input and it is more natural to use a pen to write than to use the finger. The pen is what most people use to handwrite since their infancy. Most of the gadgets built for entertainment purposes will possibly continue to follow the touch tendency. However, the author believes that professional tools, specially the ones for areas that involve handwriting recognition, will possibly tend to explore more the pen-based approach.

Finally, it has to be said that one of the main parts of these technologies is recognition. Be it handwritten text, mathematics, gestures or objects placed against the computer's screen, all of them have to be recognised so that they can be manipulated. In the context of this project, the recognition of handwritten mathematical expressions is a major technological problem. A suitable recogniser that can be used within the class library still has to be found. Not only are most of the recognisers currently available proprietary software, but also none of them seems to supports all the desired symbols and mathematical constructions. Most systems support the mathematics that is most used in school and/or the mathematics used in a particular area for which they were created.

CHAPTER 3

The functional design

In this chapter, the functional design of the *MST library* and the decisions that were made are provided. Since the main goal of this library is to teach algorithmic problem solving, each decision was taken with this in mind but also with care to make the system useful for other general mathematical tasks. However, the needs of the lecturers to teach algorithmic problem solving were always the main concern. The ideas explored in this work were presented in a preliminary stage at the "'38th ASEE/IEEE Frontiers in Education Conference" [Men08] and were received with enthusiasm by both the reviewers of the paper and the audience of the presentation. The ideas were also well received by the participants in the testing and evaluation of this project (see chapter 6 for more details).

The following decisions were made before the start of the project:

- the system will be oriented for teaching algorithmic problem solving;
- the software will be a Tablet PC application;
- handwriting will be used as the main input format.

These decisions can not be changed as doing so would be against the requirements of the project.

3.1 Library versus standalone editor

The final product of this project could be made available in two different ways:

- as a standalone application — i.e., a ready-to-use structure editor that can be used immediately by the users;
- or, as a class library — i.e., a collection of classes that can be used to extend existing ready-to-use tools to support handwritten mathematics manipulation.

As this project aims at supporting the teaching of algorithmic problem solving, creating a tool tailored to its main group of users could restrict the use of its facilities by other groups — the library's facilities can be useful in other environments and, as such, their use should not be limited to a single tool. Also, having in mind that this project was done in a research environment, it would be difficult to create a professional tool. Furthermore, the aim of research is to provide results that can be used and further developed by others. For these reasons, the main product of this project should be a class library that can be easily used by existing tools and extended if needed. This way, its features can be used in different environments and they can be adapted to the needs of its users. In order to make it easy for the library to be extended and used by others, it should be clearly structured. One way of doing that is to organise the library into modules where each module supports a set of features that are related and where each module is independent from the others. This way, programmers can extend or use only the modules whose features are of interest for them.

Decisions:

- The system should be provided in the form of a class library.
- The library should be organised in modules that can be used independently.

3.2 Supported notations

As this library is aimed at supporting the mathematics used in the teaching of algorithmic problem solving, simple arithmetic and logic expressions should be supported. Also, it should support the calculational proof format shown in section 1.2 and the writing of algorithms using the *Guarded Command Language* (for an example, see figure 1.1). It should also be possible to define new operators and redefine their properties, like their kind (infix, prefix, postfix, right and left) and precedence. This allows the system to be adapted to the user's needs.

3.3 Typeset versus handwriting

Most existing structure editors receive the input through the use of the keyboard or mouse and manipulate the typeset information. However, input using these standard devices can be a difficult task when it concerns mathematics. A more natural way to input mathematics is to handwrite it. However, handwritten input still has some drawbacks.

In a typeset environment, the input is controlled and precise, since there are only a finite number of keys or symbols for input and each symbol has a precise representation. As a result, when a key or combination of keys are pressed, the input is precise and clear, which facilitates the task of creating structure. On the other hand, in an environment where handwriting is the input of choice, the input is less precise, i.e., the same input can be written in many different ways by different users (due to differences in users' handwriting). This makes it more difficult to identify the input and create the structure that represents it. Even with these problems, handwritten input can be beneficial and thus the decision was to use handwritten input instead of normal typeset since the aim of the system is to support teaching and, in a teaching environment, having the possibility to write naturally and easily is very important in order to avoid interruptions in the users reasoning and in the lectures.

Having a mixture of handwriting with typesetting was also considered. However, as this system is to be used on Tablet PCs, it is difficult to have typeset input since there is no physical keyboard available in the tablet mode. In some platforms, a virtual keyboard is available, which can be convenient for small editing tasks, like editing recognition results returned by a recogniser. However, using it regularly for input is too tedious since one tap on the screen has to be done for each key to be pressed — which clearly takes time and is not very productive.

Decisions:

- Handwriting should be used as the main input method.
- For small editing tasks, a virtual keyboard can be used.

3.4 Handwriting recognition

As in this system mathematical expressions are handwritten, they have to be recognised before structure can be created. For this task, an existing recogniser should be used (since this project's aim is not to create recognition facilities). At the start of this project, there were not many recognisers available for handwritten mathematics that could be used within this project. Most were proprietary software and no permission was given to use them. The only company that gave access to their recogniser was *Maplesoft*. For that reason, throughout this project, this is the recogniser used. As their recogniser is for individual symbols, it does not take into account the context of what is being written and returns only the combination of the recognition results for each symbol — which obviously returns a wrong recognition most of the times. As this is a major setback for the final product, the development of the system has to take into account that the recognition of mathematical formulae is evolving very rapidly and that recognisers that are more suited for the system may become available at anytime. Thus, the system should be able to easily make use of new recognisers.

Decisions:

- Recognition should be done by existing recognisers.
- The system should allow straightforward use of different recognisers.

3.5 Text editing versus structure editing

The most commonly used approach to edit documents is the text model. This model consists in editing plain text by positioning the cursor in the position of the document that is going to be edited, and adding the text by using a keyboard or other input device that allows text input. Most computer users are familiar with the text model as it is easy to use and simple to understand. However, to write mathematics, this approach is very complicated. Mathematical expressions do not consist merely of plain text. They can be very complex and errors can easily happen using the text model. Thus, to write mathematical expressions, the text model is not appropriate.

The structured editing model consists of having the information about the structure of the document's content and allowing the user to edit the structure according to certain rules. Many structure editors only accept input defined by a fixed set of structure elements and rules. Using them is often difficult since the user has to have previous knowledge of the structure that is accepted. In spite of this difficulty, structured editing can be very useful and beneficial. For mathematical expressions, for instance, having the information about the structure makes it much easier to detect errors, to copy and edit formulae, and to perform more complex operations like applying algebraic rules. Clearly, there are still some drawbacks:

- the expressions supported are limited by the editor;
- it is very difficult to support every single option for notation, i.e., a single expression can be written in several ways and that is difficult to support.

However, the advantages for writing mathematical formulae compensate these drawbacks. For instance, selection of content is made easier and errors become more difficult to make — since the system will complain if given an expression that does not make sense in mathematical terms.

As the goal of the *MST library* is to support mathematical writing, the structured editing model should be used to deal with mathematical expressions. Nevertheless, writing mathematics that is not recognised should still be possible, i.e., it should be possible to write it in a free form environment (like if it was a drawing). However, in this case, there will not exist a structured representation of the formula and thus, it will not be possible to apply the structure manipulation features provided by the system.

Decisions:

- The structure editing model should be used for mathematics.
- It should be possible to input handwritten mathematics in a free form environment.

3.5.1 Basic editing operations

When writing mathematics, content is constantly being copied. Frequently, from one step of a calculation to the next one, only small changes are made to the expressions. Thus, one has to repeat things that were already written, when they could simply make use of a copy operation and edit solely the parts of the expression that need to change.

Having a copy operation saves time for the user and also avoids the introduction of errors, which often happens when copying content. Hence, the system should provide a copy operation that is, preferably, simple and natural to use, since it is used often. Also, to copy expressions, a structured selection operation has to be available. In order to copy content effectively and accurately, the selection should also be accurate, easy to use, and follow the expressions' mathematical structure. This selection feature is also needed to allow the use of manipulation rules (see section 3.6).

Decision:

- Structured selection and copy facilities that are simple and natural to use should be provided.

3.6 Manipulation rules

In order to help with calculations and to show the dynamics of algorithmic problem solving, the library should provide the facilities to apply some algebraic rules to handwritten expressions. The rules that are most often used should be supported — like symmetry, distributivity and substitution of equals for equals (*Leibniz*). The result of applying a rule could be provided either in typeset or handwritten representation. However, mixing typeset with handwriting can be confusing (since it is more difficult to see which expressions are related due to their differences in representation) and it does not keep the environment similar to teaching on the blackboard. For these reasons, the result of applying an algebraic rule is provided in a handwritten representation. Given that some rules introduce symbols that are not present in the original expression, it should be possible to define the user's handwritten representation of characters in order to use them whenever it is needed.

Decisions:

- It should be possible to apply algebraic rules to handwritten expressions.
- It should be possible to define and save user's handwritten characters.

3.7 Animating manipulation rules

Having in mind that this system is aimed at teaching mathematical content, some features should be provided to help showing to the students how calculations are done. In particular, it is very important that students can understand how algebraic rules are used. In the author's experience, many students find it difficult to apply algebraic rules because they have problems with finding which rules can be applied to expressions — since sometimes it is not immediate to see that the shape of an expression is the same of a certain rule. For example, it can be difficult for some students to see that the factorisation,

$$a \times b + a \times c = a \times (b + c),$$

can be applied to the expression

$$a \times b \times c + d \times b \times c \times e$$

since the shape of the expression does not match exactly the shape of the rule. By using associativity and symmetry, the two expressions can become more similar and thus it will be easier to see that their shape is the same. Having a system that lets the students "play" with the expressions and that applies the rules for them, can help the students with finding how rules can be applied. In this context, the support for manipulation rules is important. However, when rules are applied within this system, the final result of applying them appears almost immediately on the screen. This rapid change of the original expressions to the result may be difficult to follow which can still make it difficult to understand how the rules work. For this reason, there should be a way for students to see how the expressions were transformed in order to get the final result. Ideally, students should be able to see how the parts of the expressions were used to obtain the final result. This can help them understand how it works. Thus, it would be good to have a way to see the several steps of the transformation of the expressions when a rule is applied.

Decision:

- It should be possible to see a step-by-step animation of how the rules are applied to expressions.

3.8 Use of gestures

One of the goals of this project is to provide a straightforward, flexible and also reliable way of writing and editing mathematics in a computer. The final system has to allow fluidity of thought and easy editing/manipulation of handwritten mathematics. For that, it should be easy to trigger editing tasks and manipulations. Using toolbar buttons to start these actions can distract the user from what they are doing, since their focus will change to other areas of the screen. Also, toolbars are usually small and can be difficult to target with a pen. To avoid distractions, the actions that are often used should be triggered in a straightforward way, preferably in the area that the user is editing. For that, *gestures* can be useful. Gestures are specific pen motions that can initiate commands. For instance, a quick circle (see figure 3.1) may select the ink situated inside it. Gestures are usually inked in the area of the screen that the user is editing. Thus, gestures can help with keeping the user focused and should be used to help achieve a system that interferes as little as possible with the user's thought.

Decision:

- Gestures should be used to trigger basic editing tasks and structured manipulations.

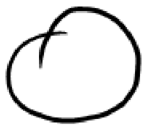


Figure 3.1: Circle gesture

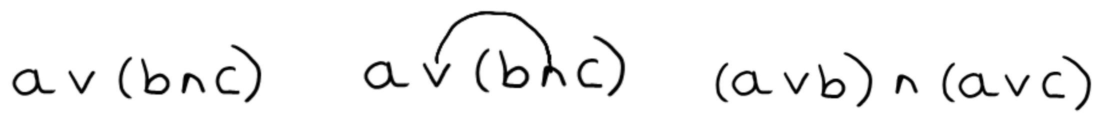
3.8.1 Gestures to trigger actions

The system should support the user when performing certain mathematical operations. Those operations will be triggered by the user and the user is responsible for the correctness of applying them. The gain for the user is that they do not have to determine themselves the result of applying an operation and errors are more difficult to make. This can improve the simplicity and accuracy of mathematical calculations. As mentioned above, to make things as simple as possible, gestures should be used to trigger some rules that are used often. An example of such a rule is the distributivity rule. Consider the distributivity of Boolean disjunction through conjunction, that is:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \ .$$

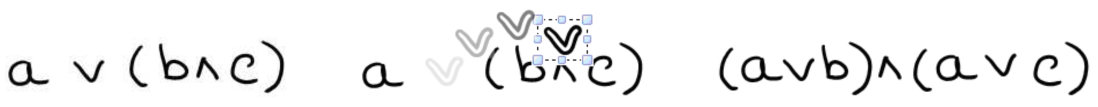
If the user wants to apply the distributivity rule to an expression of the shape of the left-hand side of the rule, they should have the possibility to do that with the support of the library. A gesture will indicate that the rule should be applied. Associations between gestures and rules can be customised by the user. By default, the system will provide, for each task, gestures that seem intuitive for that particular task. For example, two candidate gestures for applying the distributivity rule are the following:

- draw a semi-circle starting at the $\vee$ symbol and ending on the $\wedge$ symbol (figure 3.2);
- select the $\vee$ symbol, drag it and drop it over the $\wedge$ symbol (figure 3.3).



(a) Expression **(b)** Gesture **(c)** Result

Figure 3.2: Using the semi-circle gesture to apply a rule



(a) Expression **(b)** Gesture **(c)** Result

Figure 3.3: Using the drag-and-drop gesture to apply a rule

Both gestures match the visual idea that one operator will be propagated over the other, making then natural candidates to trigger the distributivity rule.

However, one should be careful with the use of gestures since too many gestures can interfere with each other and with the user. For instance, the circle gesture mentioned above can interfere with the writing of number zero or of the character 'o'. Another example is that the semi-circle gesture may interfere with drag-and-drop. If drag-and-drop is done as it is shown in figure 3.3, the pen follows the same path that is done to write the semi-circle. That can trigger the semi-circle action instead of the drag-and-drop. Thus, care should to be taken to make sure that actions are not inadvertently triggered.

To conclude, it has to be said that, in general, inking environments do not have the notion of cursor (as in typesetting environments). So, another reason why gestures should be supported is that they can be used to indicate the location where a certain action should occur.

3.9 System versus user in control

Many tools try to take control of what the user is doing. Tools can, for instance, show to the user that certain mathematical manipulations are not correct. But given that there are many different notations to represent the same mathematical result and that new notations and meanings can be created at any time, it is best that the system does not try to impose any mathematical definitions. For these reasons, the system should only have the general knowledge of mathematical structures and the knowledge of how to apply rules given a structure. As long as the rule can be applied to a certain structure, it will be applied without interpretation of the expression and without making sure that the step is valid. Assuring the validity of the steps has to be the user's responsibility. Also, for the same reason, the system should allow the user to create new operators and redefine the operator's properties (kind and precedence).

As mentioned above, gestures are used for editing operations and to trigger actions. Gestures are usually easy and straightforward to use, but for some users, some gestures can seem less natural. For this reason, the relation between gestures and actions should not be fixed. The user should be able to associate gestures with certain actions in the way they find best suited for their needs. The gestures used by default should be defined taking into account interferences between gestures and also the ease of use. From the moment that the user starts changing the default definitions, it is their responsibility to detect and control interferences between gestures.

Finally, the user should be able to edit recognition results. Some systems assume that the recogniser is perfect and that the returning result is the one to be used. However, as this is clearly not the case in mathematics recognition, the user should be free to choose the appropriate recognition result from the options that the recogniser returns or, in case no result matches the input, the user should be able to input the correct recognition.

Decision:

- The user should be in control and not the system:
    - it is the user's responsibility to check semantic validity;
    - it should be possible to redefine the gestures associated with certain actions;
    - it should be possible to create new operators and redefine their properties;
    - it should be possible to edit recognition results.

3.10 Combined writing of mathematics and text

To present mathematics effectively, it is necessary to present the mathematical results accompanied by some explanation. For that, the combined writing of text and mathematics is needed. As this system aims at presenting mathematics, it is desirable to support both handwritten mathematics and text, but it is not crucial to provide special facilities to deal with text (like the ones available in standard text editors). Text can be seen just as ink (like a drawing) that can be written and deleted as desired.

Decision:

- The system should allow combined writing of mathematics and text, but it is not necessary to provide structured editing of text.

CHAPTER 4

The technical design

In this chapter, the technical design of the *MST library* is presented. In the next few sections, the most important features are described and details of their implementation are given.

The final product of this project is available in the form of a C# class library oriented for Tablet PCs. This library was created using *Microsoft Visual Studio 2008[vis12]* and it makes extensive use of the *Microsoft Tablet PC platform SDK v1.7 APIs*[tab12]. This SDK is the standard development kit for producing ink- and pen-enabled Microsoft Windows applications.

4.1 Architecture

The general architecture of the system is depicted in figure 4.1. In this diagram, all the actions that are triggered by a specific action are written in the same colour. The *user interface* allows the user to input handwritten text and mathematics, and manipulate the mathematical expressions. First, the input is sent to the *strokes handler* which divides the ink into collections of strokes, each representing a character. The divided strokes are then passed on to a *character recogniser*[1] which provides, for each symbol, a set of recognition results. Each result is a textual representation of the handwritten expression. One of the recognition results is then selected by the user as the correct one. After the selection of one recognition result, the structure that represents the handwritten expression is created. That is the job of the *parser*. If the recognition result is a well-written expression, then it will create an internal representation of the input and

[1]In this thesis, we use the terms *character* and *symbol* interchangeably.

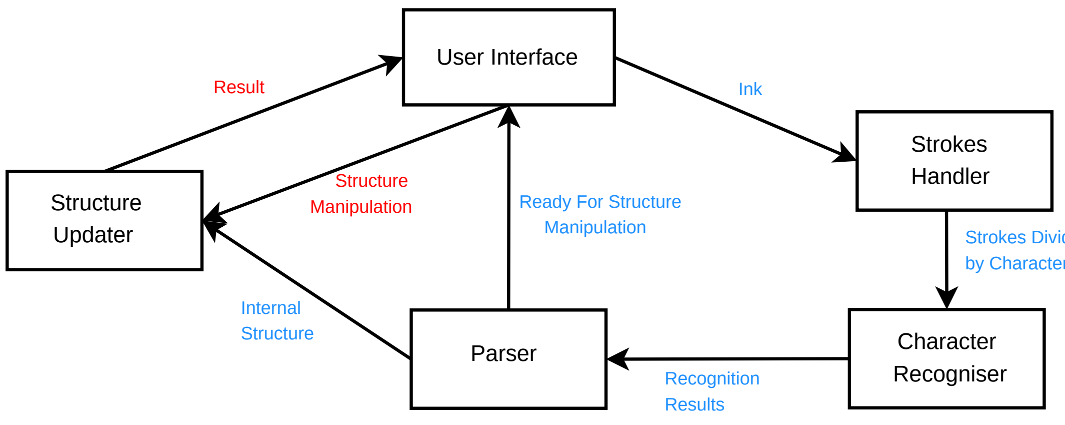


Figure 4.1: System architecture

allow the user to manipulate the expression. Every time the user manipulates the expression, the *structure updater* will update its structure (it will modify the expression's tree according to the manipulation that was done).

It should be noted, however, that the *user interface* presented in the diagram is not part of the *MST library*. It is part of some program that uses this library and knows how to communicate with it — as will become clearer in the remainder of this thesis.

4.2 Extendability and reusability

As the use of Tablet PCs and the recognition of handwriting have attracted much attention in the last few years, things tend to change rapidly. Thus, a system that is created today can become outdated in just a few years time. To avoid that, one of the goals of this project is to provide an extendable library. One of the main concerns is that new recognisers with much better recognition capabilities can appear; in that case it would be desirable to use the new ones instead of a fixed recogniser. For this reason, in the architecture diagram presented in figure 4.1, the *character recogniser* and the *parser* do not form an unique block. The idea is to allow the connection to the system of new recognisers and new parsers, or even a block that can work both as a recogniser and as a parser. For that, this library was created to accommodate change by the introduction of the so-called *strategy pattern* [GHJV95]. The *strategy pattern* is a *design pattern*, that is, it is a general reusable solution to a commonly occurring problem in software design.

Design patterns describe problems which occur often, and describe a simple and elegant solution for those problems in a way that they can be reused over and over again. There are many types of *design patterns*. For a detailed catalogue of these, the book [GHJV95] is recommended.

The goal of the *strategy pattern* is to define a family of algorithms, encapsulate each one, and make them interchangeable. This pattern is useful in applications that require dynamic swap of algorithms (in this case, several recognisers can be used with the library), but the way it is defined is also useful to easily connect new recognisers and parsers to the library. In figure 4.2, a UML diagram of the strategy pattern as it is used in this system is presented.

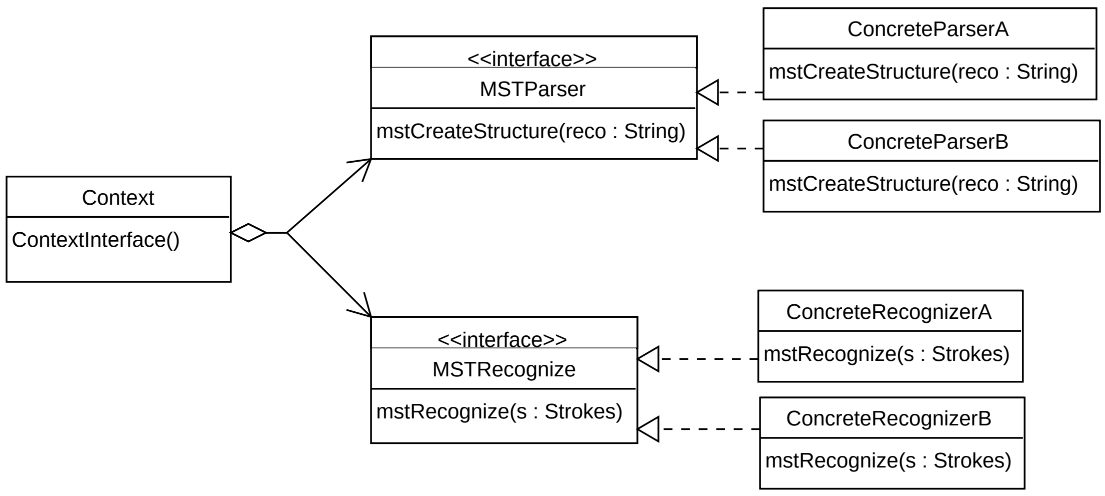

ConcreteParserA
mstCreateStructure(reco : String)
<<interface>>
MSTParser
mstCreateStructure(reco : String)
ConcreteParserB
mstCreateStructure(reco : String)
Context
ContextInterface()
ConcreteRecognizerA
mstRecognize(s : Strokes)
<<interface>>
MSTRecognize
mstRecognize(s : Strokes)
ConcreteRecognizerB
mstRecognize(s : Strokes)

Figure 4.2: Usage of the Strategy pattern

To add a new recogniser, the interface `MSTRecognize` has to be implemented by that recogniser. This interface defines a single method `mstRecognize` which is used to recognise mathematics. Similarly, to integrate a new parser with the tool, the interface `MSTParser` has to be implemented. The method `mstCreateStructure` is used for the parsing of the mathematical expressions.

The use of the strategy pattern makes it not only easy to integrate new recognisers and parsers, but it also allows the use of more than one recogniser in runtime.

4.2.1 Reusability

During the development of the *MST library*, special attention was given to its reusability. It is desirable that the system can be used within other tools, that it can be adapted to new developments, and that it can serve as a base for the development of new features. For these reasons, the system is made available as a library and it was created in a modular way — that is, it is organised in modules, each one supporting a main feature of the system. The library is made up of four main modules[2] , namely, the modules *Recognisers*, *Parsers*, *Structure* and *Gestures*. As their names indicate, the module *Recognisers* contains the recognition facilities, *Parsers* contains all the code for the parsing of expressions, *Structure* contains all the definitions for mathematical structures and the methods that manipulate them, and, finally, the module *Gestures* gathers all the facilities that deal with gestures. In the remainder of this chapter, many of the features included in these modules are detailed. For details on how to use these modules within other tools, chapter 5 is recommended.

4.3 Handling of strokes

In order to create structure for mathematical expressions, the mathematical handwritten input has to be recognised. Given that the library assumes the use of a symbol recogniser, the strokes have to be divided in collections of strokes, each collection containing all the strokes that form a symbol. Assuming that symbols are composed only by strokes that intersect, an approach to the division of the strokes into collections of strokes is based on determining recursively which strokes' bounding boxes intersect which. Every group of strokes that are part of the intersection's closure for one initial stroke, are part of the same collection. For example, consider the character 'H' in figure 4.3, which composed by three strokes: *s1*, *s2* and *s3*.

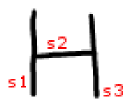


Figure 4.3: Handwritten character 'H'

To determine the strokes that make part of this character, the following algorithm can be followed:

[2]Note that, however, there are other auxiliary modules. For example, there is a module called *IO* which allows the writing of markup code to represent mathematical expressions (its features are described in section 4.10). There is also a module *Stencils*, which provides features for creating handwritten templates (for more details see section 4.11.1).

1. start a collection (*result*) with the stroke *s1*;
2. find all the strokes whose bounding boxes intersect the bounding box of *s1*: *s2* is found;
3. *result* does not contain *s2*: add *s2* to the collection;
4. find all the strokes whose bounding boxes intersect the bounding box of *s2*: *s1* and *s3* are found;
5. *result* contains *s1* but does not contain *s3*: add *s3* to the collection;
6. find all the strokes whose bounding boxes intersect the bounding box of *s3*: *s2* is found;
7. *result* already contains *s2*: stop the search;

This strategy works well when dealing with characters where all their strokes touch each other. However, for characters such as 'i', the above strategy ends up returning two collections: one with each stroke. That is no good since they are sent to the recogniser as two different characters, and thus they are recognised separately which will never return the character 'i'.

To overcome this problem, the algorithm can be improved by searching for the intersections by using extended bounding boxes. For each stroke, when trying to find which other strokes intersect its bounding box, the bounding box used is extended up and down to the size of the line that the stroke in question is part of. Figure 4.4 illustrates the extension for the case where an 'i' is being processed. The blue rectangle corresponds to the bounding box that contains the whole line. The pink one is the bounding box of the main stroke that forms the 'i' character. Using only this bounding box would let the dot that is part of the character outside the collection. However, to correct that, the bounding box is extended with the red rectangle, that is, the new bounding that is going to be used is the bounding box composed by both the pink and the red rectangles. That corresponds to changing the height of the pink bounding box to the height of the line (the blue rectangle) and change the y-coordinate to be the y-coordinate of the line. The algorithm that extends the bounding boxes is the one used in the *MST library*.

It should be noted, however, that in the current implementation, only strokes that intersect, or that are situated above or below each other, are considered part of the same

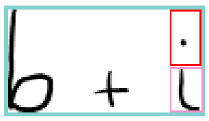


Figure 4.4: Extending the bounding box of 'i'

character. For example, the system fails to divide correctly the strokes that form the assignment symbol (:=), since the strokes of the colon do not intersect the strokes of the equals sign. Users need to have this in mind when writing input that will be consumed by the library.

Before being divided into collections of strokes using the above algorithm, the strokes being processed need to be divided by lines. *Microsoft Tablet PC API* provides the *Ink.Divider* class that allows the division of the ink into drawings, lines, paragraphs and segments of recognition. Initially, the *MST library* performed line division using the *Ink.Divider* class but the results were very unreliable. For this reason, a new division of the ink by lines was implemented. The new division, although not perfect, gives the correct result more often than the *Divider* for the kind of mathematics that this work supports (please see section 6.4 for more details). The algorithm for this division is algorithm 1.

Each iteration of the loop adds a line to the array *lines* and removes it from the argument *strokes*. Because the size of *strokes* is decreased at each iteration, the algorithm terminates. The method *sortByY* sorts a collection of strokes in ascending order of their y-coordinate. Method *determineIntersections* determines which strokes intersect a given rectangle (in this case, the rectangles used are *lineBB* and the bounding box of *firstLine*). The repeated call to *determineIntersections* is used to improve accuracy: by calling the function again with the bounding box obtained in the previous iteration (*lineBB*), the algorithm may expand the line to contain strokes that were not initially intersected. To better appreciate how the algorithm improves, consider the input "2 ; g"; if the repeated call to *determineIntersections* is not used, *firstLine* becomes "2 · g".

Algorithm 1 GetLines: Dividing *strokes* by lines

```
Require: Assume that the stroke with the smallest y-coordinate is part of the first line;

lines := Empty;
while strokes $\neq$ Empty do
  sorted := sortByY(strokes);
  firstLine := sorted.First;
  repeat
    nStrokes := firstLine.Count;
    lineBB := newRectangle(strokes.X, strokes.Y, strokes.Width, firstLine.Height);
    firstLine := determineIntersections(lineBB, strokes);
  until nStrokes == firstLine.Count;
  strokes.Remove(firstLine);
  lines.Add(firstLine);
end while
return lines
```

4.4 Character recogniser

Although it is possible to change the recogniser that is used with the library (as described in section 4.2), the recogniser that was used during the development of the library was a symbol recogniser developed by Maplesoft [map12]. This recogniser is not the ideal for this task since it does not take into account any context of the expressions and solely recognises each symbol individually. To obtain good results, the recognition should be done with information about the context in which the individual symbols are being written. For that, probabilistic models can be used. A short description of these models, is presented in section 4.4.1.

As the `Maplesoft`'s recogniser is written in `JAVA` (and this project is written in C#), a script is used to make a system call to run the recogniser and send to it the collections of strokes[3]

After the division of the strokes into single characters, the collections of strokes are

[3]There is also a version that uses Maplesoft's recognition through a DLL. There were many difficulties in creating the DLL file — mainly because of the lack of knowledge of how the recogniser is written — but after many attempts, a DLL was created and seems to be faster than the use of system calls. However, the results do not seem as accurate.

sent one by one to the recogniser. For each character, the recogniser returns a list of recognition results. The *MST library* combines these lists to produce a list of recognition results for the entire expression. These results are then shown to the user using a menu similar the one presented in figure 4.5.

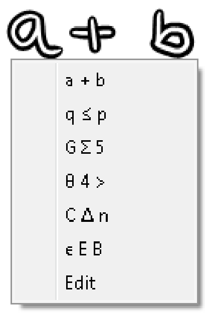


Figure 4.5: Context menu with recognition results

In this example, the results that were returned for the character 'a' were 'a', 'q', 'G', 'θ', 'C' and 'ϵ', in this exact order. That is why these are the results that appear in the first six results presented for the whole expression (similarly, the first six results for '+' and 'b' are the ones shown in the menu). Other strategies to combine the results can be implemented.

The user is presented with a menu like the one shown in figure 4.5 so that they can choose the correct recognition. If the correct recognition is one of the choices presented, then the user has to select the correct one. If the correct result does not appear in the menu, then the 'Edit' button should be clicked and the correct recognition should be written manually by the user.

After this step, the string that represents the input is obtained and can be parsed to create its structure.

4.4.1 Bayesian network models

In the handwriting recognition field, it is common to use probabilistic models to embed the information needed for the recognition and base the recognition results on probabilities. In this project, a recogniser for individual characters is used together with an ANTLR parser to create the structure for the set of symbols that were recognised. However, it might happen that an individual symbol is recognised wrongly. As an example, a '$+$' can be recognised as a 't'. So, if one writes '$1 + 2$' and the characters '1', 't' and '2' are sent to the parser, the parser does not know how to behave since there is no rule for this expression (in a mathematical context). However, if we had a probabilistic model to help us with the recognition, it would know that between two numbers it is more likely to have an operator than a letter. That is why it was considered using probabilistic models in this project to improve the recognition results and to make the recognition process simpler for the user. In this context, *probabilistic graphical models* were considered. These models were used with success, for instance, in [Htw07] for recognition of handwritten Pitman's shorthand. A *probabilistic graphical model* is a graph that represents probability distributions and *Bayesian networks* are one of the most important classes of graphical models. These networks are directed acyclic graphs in which the nodes represent variables and the arrows represent causal relationships between the variables — they represent probabilistic dependencies between the variables. An arc from a node A to a node B indicates that A 'causes' B. Bayesian Networks are so called because they use Baye's rule for probabilistic inference [Mur98, Pea85, Pea88]. Using these networks, it is possible to determine which symbol is more likely to follow a given symbol or set of symbols. To use this in this project, the graph structure and the parameters of the model would have to be defined (e.g. the conditional probability distribution at each node).

There are algorithms to perform inference and learning in Bayesian networks which would be very useful to improve the recognition results. However, the recognition of handwriting was not planned as a task of this project and to implement a Bayesian network would take considerable amount of time. Thus, this possibility was abandoned and left for a future improvement of the library.

In the *MST library*, to avoid errors, the user has always to verify if the recognition returned is correct, and edit it if that is not the case. The string with the recognition result can only be sent to the parser after a correct recognition result is obtained. With a Bayesian network this step could possibly be avoided, making the tool much easier to use.

4.5 Parsing recognition results

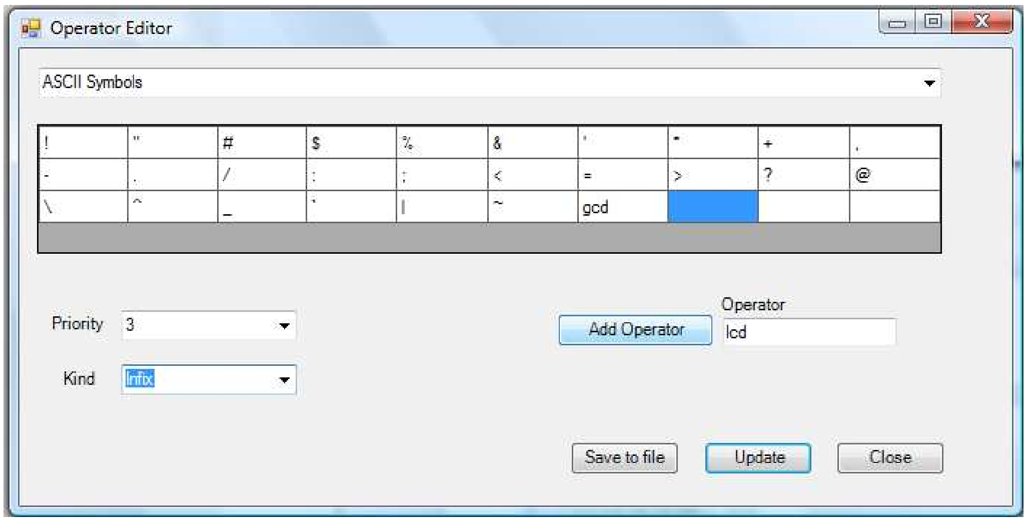
After a recognition result is obtained, it is parsed by a parser which was generated by *ANTLR* [ant12]. Given a grammatical description, *ANTLR* generates a parser. More specifically, it generates a lexer and a parser: the lexer takes a stream of characters and divides the stream into tokens according to pre-set rules; the parser reads the tokens and interprets them according to its rules. In this thesis, when the term parser is used, it refers to the set of generated files, that is, to the lexer and the parser. The expressions “parser rule” or “lexer rules” are used to refer to the rules defined in the grammar file for defining, respectively, the generated parser and lexer.

To parse a given recognition result, the string that is sent to the parser contains the recognition result together with the information of the bounding boxes of the strokes that compose each character. An example of these strings is the following:

```
"4366 3168 3518 949 a 4366 3541 1002 576 + 6031 3496 689 474 b 7323 3168 561 835"
```

This string represents the expression $a + b$. The first four values correspond to the x and y coordinates, and to the width and height of the bounding box that contains all the strokes that form this expression. After this, comes the character ’a’ and the values that represent the bounding box that contains all the strokes that form this character. The same applies to the characters ’+’ and ’b’. The parser reads this string and creates a tree that represents this expression.

As one of the objectives of this project is to provide a library that can be adapted as much as possible to the user’s needs, and given that in mathematics there are many different notations for the same operators and new operators can be created at anytime, this library allows the addition and redefinition of operators in runtime. For that, an operator editor (see figure 4.6) is provided and the parser was created as a dynamic parser that parses any new symbols that the users add to the operators list.



ASCII Symbols

!	"	#	$	%	&	'	*	+	,
-	.	/	:	;	<	=	>	?	@
\	^	_	`	|	~	gcd			

Priority 3

Kind Infix

Operator lcd

Figure 4.6: Operator Editor

The editor provides an interface to add new operators to the system and change their properties (kind and priority). In figure 4.6, the user is adding an operator named *lcd* with kind *infix* and priority *3*.

To make the parsing dynamic in terms of the operators that it supports, specific features of *ANTLR* are used. More specifically, *ANTLR* allows the change of the type of a lexer token dynamically. The following lines illustrate this feature:

```
ID   :    ('a'..'z'|'A'..'Z')+
                  { if(isBOperator($text)) $type=DYNAMICBOP;
                    else if(isPREOperator($text)) $type=DYNAMICPRE;
                    else if(isPOSTOperator($text)) $type=DYNAMICPOST; } ;

UNICODE : '\u0000'..'\uFFFE'
                  { if(isBOperator($text)) $type=DYNAMICBOP;
                    else if(isPREOperator($text)) $type=DYNAMICPRE;
                    else if(isPOSTOperator($text)) $type=DYNAMICPOST; } ;

fragment DYNAMICBOP: ;
fragment DYNAMICPRE: ;
fragment DYNAMICPOST: ;
```

`ID` and `UNICODE` define tokens that contain, respectively, one or more characters between 'a' and 'z' (upper and lower case), and all the characters in the unicode range from '0' and 'FFFE'. The "if .. else" statement in each of these rules checks whether the input to be consumed is a binary operator (*isBOperator*), a prefix operator (*isPREOperator*), or postfix operator (*isPOSTOperator*) and modifies its type accordingly (to `DYNAMICBOP`, `DYNAMICPRE` or `DYNAMICPOST`). The lines that start with the word *fragment* instruct ANTLR that the rule is only used as part of another lexer rule. In this case, the three fragments defined are used as part of the lexer rules `ID` and `UNICODE`. This approach allows to determine if, in the current context, a token is an operator. For example, the *isBOperator* method is defined as follows:

```
private Hashtable opInfo = MST.MSTVariables.OpToOpInfo;

Boolean isBOperator(String s)
{
    if(opInfo.ContainsKey(s))
    {
        if(((OperatorInfo)opInfo[s]).Kind < 3)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
```

Three types of binary operators are considered: infix (associative), left-associative and right-associative. These types are identified by the number 0, 1 and 2, respectively. The methods *isPREOperator* and *isPOSOperator* are defined very similarly, changing only the kind involved in the test — the kind for prefix operators is 3 and for postfix is 4.

The *OpToOpInfo* that appears in the code is a global variable defined in the system in the *MSTVariables* class. This class is used to define all the system variables. The *OpToOpInfo* is a *Hashtable* that associates operators (string) with an object of type *OperatorInfo*. The *OperatorInfo* class has the following definition (shown partially):

```
public class OperatorInfo
{   private int precedence, kind;

    public OperatorInfo(int prec, int k)
    {
        this.precedence = prec;
        this.kind = k;
    }
    (...)
```

Thus, an *OperatorInfo* can be seen as a pair that holds the precedence (or priority) and the kind of an operator.

Parser rules: The start rule of the parser is *complete_expression* which merely states that the input should be of the form x-coordinate (x), y-coordinate (y), width (w) and height (h) followed by an *expression* (e). Both the *complete_expression* and the *expression* rule are defined as follows[4]:

```
complete_expression:
    x=INT y=INT w=INT h=INT e=expression EOF;

expression returns [Node local]:
    (    left=primary_factor;
    )
    (    b=DYNAMICBOP x=INT y=INT w=INT h=INT
         right=primary_factor;

         | b=DYNAMICBOP x=INT y=INT w=INT h=INT
           ('\U23A8'|'ht') x1=INT y1=INT w1=INT h1=INT e=expression
           ('\U23AC'|'th') x2=INT y2=INT w2=INT h2=INT
           right=primary_factor;

         | right = primary_factor;
    )*
;
```

As the code above shows, an *expression* is composed by:

- a *primary_factor*;
- or, a *primary_factor* followed by one or more occurrences of:
  - a binary operator and another *primary_factor*;
  - or, a binary operator followed by an *expression* between the tokens '\U23A8' (or 'ht') and '\U23AC' (or 'th') which, in turn, is followed by another *primary_factor*;
  - or, another *primary_factor* (this is the case where there is an invisible operator between two *primary_factor*s);

[4]The C# code was removed from the rules to make it more simple to read the rule's definitions.

The case where an operator is followed by an expression between the tokens '\U23A8' (or 'ht') and '\U23AC' (or 'th') is used for writing calculational proofs. The tokens are used to delimit the hints. Although hints in calculational proofs are usually presented between curly brackets as in

$$\begin{array}{ll} & A \\ = & \{\quad \text{hint} \ \} \\ & B \end{array}$$

the current system is using the special keywords '\U23A8' (or 'ht') and '\U23AC' (or 'th'). This restriction comes from the fact that invisible operators and expressions delimited by curly brackets are permitted. Since invisible operators have the lowest precedence, if the curly brackets were used to delimit hints, the expression $A = \{hint\}B$ would be read as $(A = \{hint\})B$ since it would assume an invisible operator between B and the rest of the expression. For this reason, a different type of brackets has to be used to delimit hints. The unicode characters chosen are very similar to curly brackets, as figure 4.7 shows. The 'ht' and 'th' tokens are provided since they are ASCII and, in case the recognition result has to be edited, it is easier to input these delimiters than the unicode versions.

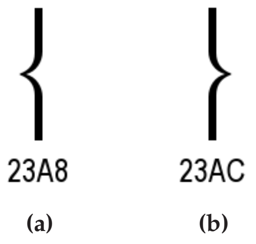


Figure 4.7: Brackets used for hints

The *primary_factor* rule used above is defined as:

```
primary_factor returns [Node local]:
    pre=DYNAMICPRE x=INT y=INT w=INT h=INT pe=primary_expression;

    | pe=primary_expression f=factor;
```

A *primary_factor* is either a prefix operator followed by a *primary_expression* or a *primary_expression* followed by a *factor*. As can be seen from its definition, a *factor* is either a postfix operator or the empty string:

```
factor returns [Node local]:
        post=DYNAMICPOST x=INT y=INT w=INT h=INT;

        |    ;
```

Finally, the *primary_expression* defines the parsing of identifiers, unicode characters, strings, algorithms, expressions between brackets and quantifiers. The quantifiers supported are of the form:

$$\langle \bigoplus bv\colon range\colon term \rangle.$$

This notation is know as the Eindhoven quantifier notation. There are five components to the notation. The first component is the *quantifier* $\bigoplus$. The second component are the bound variables *bv*. The third component is the range of the bound variables. The range is a Boolean-valued expression that determines a set of values of the bound variables for which the expression is true. The fourth component is the term, which is an expression that will be evaluated for each value of the bound variables; the way the results are put together is determine by the quantifier (for instance, in a summation, all the results are added). For more details on this notation, [Bac03] is recommended.

4.5.1 Data structures for mathematical expressions

As already mentioned before, this library allows structured manipulation of mathematical expressions. This is possible, because the parser creates a data structure that represents the mathematical expressions. This section describe the data structure that is used.

The first data structure that was considered was a binary tree structure, because their use is very simple and they are widely known. However, these trees are not suitable for all kinds of mathematical expressions as explained in Verhoeven's thesis on Math∫pad [Ver00]. The mathematical expressions that are supported by this library are supported by the Math∫pad system. So, some of the results presented in Verhoeven's thesis are useful for this project. In the Math∫pad project, Richard Verhoeven and Olaf Weber have used *rose trees* [ros12] to represent mathematical expressions. A rose tree is a tree in which the nodes have a list of subnodes (and not just two subnodes as in binary trees). The number of subnodes is not restricted which is very useful to store the expressions. Using these trees, an expression can be divided into sub-expressions which will be stored as subnodes of the whole expression.

For the same reasons that rose trees are used in the Math∫pad project, they are also used for storing mathematical expressions in the *MST library*. However, the information stored in the nodes of the trees and the way in which the trees are created and searched are different from what is done in Math∫pad.

Each *Node* of the rose trees that are used to represent mathematical expressions contains the following information:

```
public class Node
{
    Rectangle boundingBox;
    char reco;
    int kind;
    Parent parent;
    Node first;
    Node last;
    Node right;
    Node left;

    ...
}
```

The first two variables are related with the handwritten expression:

- *boundingBox*: holds the smallest rectangle that contains the strokes to which this node refers to; a *Rectangle* is an object that is composed by four elements: x-coordinate and y-coordinate of the top left corner, plus the height and width of the rectangle;
- *reco*: holds the recognition result of the strokes that this node refers to.

The other variables are concerned with the mathematical structure:

- *kind*: holds one of the following values and represents the type of the expression that is being held:

```
int ROOT = 0;
int EXPR = 1;
int OP = 2; // Includes Infix , Prefix and Postfix , Left and Right
int ID = 3;
int TEXT = 4;
int BRACKET = 5;
int SEPARATOR = 6;
int OP_PREFIX = 7; // currently unused
int OP_POSTFIX = 8; // currently unused
int OP_NONE = 9;
```

- *parent*: holds the information that each node needs to know about its parent node. The class *Parent* has the following instance variables:

```
public class Parent
{
    Node parent;
    int position;
        ...
}
```

For a given node, the *parent* variable holds the parent node, and the *position* variable holds its position in relation to its parent. For instance, given the expression $a+b$ the tree that represents it is as follows:

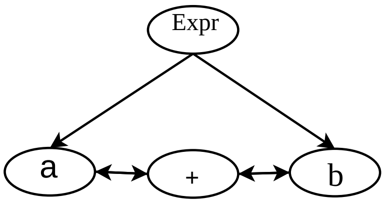


Figure 4.8: Rose tree that represents the expression $a+b$

The parent of 'a','+' and 'b' is the node *Expr* and their position in relation to the parent is 1, 2 and 3, respectively. Currently, the position information is not being used, but it was included since it was supposed to be used on the writing of text intercalated with mathematical expressions. However, the current solution has a limited support for combined writing of mathematics and text and it does not need this information (more details are presented in section 4.5.2). Nevertheless, the position information is available so that it can be used to improve the writing of text.

- *first*, *last*, *right*, *left*: hold, respectively, the first child of the node, the last child, the node that stands on its right, and the node that stands on its left.

It should be noted that the bounding boxes that are held in the nodes are using ink space coordinates. The Tablet PC Platform has (at least) two coordinate systems: pixel coordinates (or device coordinates) and ink coordinates. The difference between them is that ink coordinates are much denser than pixel coordinates, since digitizers have a much higher resolution than displays. It is important to use ink space coordinates because the structure relies on the position of the ink within the *Ink* object. If pixel coordinates were used, every time a resize of the window was done, the coordinates would change (because their coordinates in relation to the display would change) and either the whole structure would have to be updated every time this happened (which would be costly in terms of efficiency) or there would be a mismatch between the structure and the screen. *Ink* objects and *Stroke* objects use the HIMETRIC coordinate system. A HIMETRIC unit represents 0.01mm, where the measurement is derived from the screen's current DPI (dots per inch). The origin of the coordinate space is the point (0,0) and it represents the upper-left corner of the space. The consistency between the *Ink* object and the display is achieved through the *Renderer* class. To draw a *Stroke* object to the screen, a *Renderer* object obtains the ink coordinates from the *Stroke* object, transforms them, converts them into pixels, and finally renders that data on the screen. Figure 4.9 illustrates the interaction between these entities. For more details see [JS03]. It should be noted that, in general, users of the *MST library* do not need to deal with these transformations as these are done automatically.

Now, to illustrate the way trees are created, consider the following expression:

$$a + b + c \times d \times e + f \times g \ .$$

A representation of the structure that is created for this expression is presented in figure 4.10.

As the figure shows, the operators that have the same precedence are all in the same level of the tree. Different levels express different precedences. For instance, the multiplication is in a higher level than addition because its precedence is higher.

This tree structure is convenient for the structured selection, which will be detailed

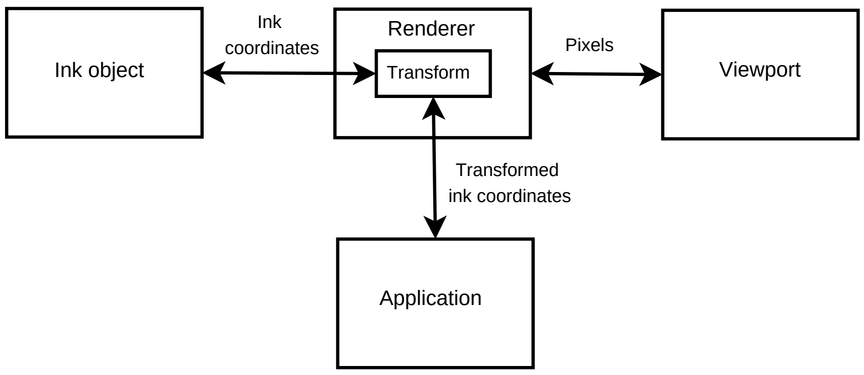


Figure 4.9: How a *Renderer* object draws ink strokes to a viewport

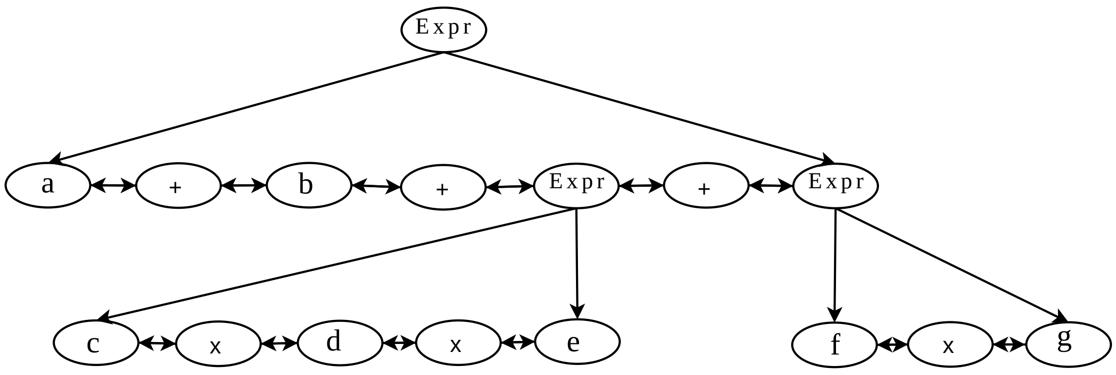


Figure 4.10: Rose tree that represents the expression $a + b + c \times d \times e + f \times g$

in section 4.7, and for applying algebraic rules. For instance, for the sub-expression $b + c \times d \times e$ of the expression presented above, if one applies a swap (symmetry) to the addition operation, the update of the structure is very simple. It consists of simply swapping the positions of the node b and the node $Expr$, as shown in figure 4.11.

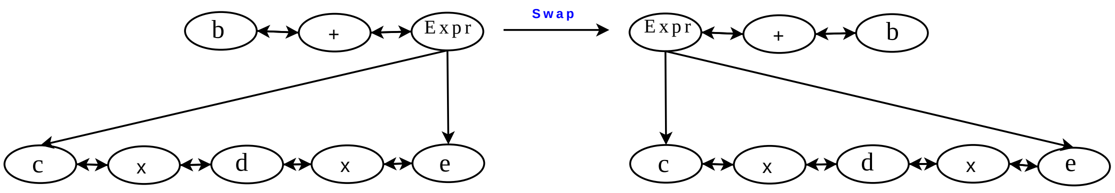


Figure 4.11: Swap operation

The tree structure that is created by the library contains some nodes denoted with $Expr$. These are *Expression* type nodes, meaning that they are the root of an expression (or sub-expression). The *Expression* type nodes are useful for making the search more efficient. These nodes contain the bounding box of the expression that they hold, which avoids searching all the nodes of the expression individually to find a certain node. To clarify this, consider that a tap is done over an element of a mathematical expression. To find out which node holds that element, the node that contains the point where the tap was done has to be found. With the use of the *Expression* nodes, the search function does not need to check all the nodes of each expression in the document. It just needs to find out the expression that contains that point, and then search inside that expression. To find the expression that contains the point, the only operation that needs to be done is to check whether that point is inside the bounding box of the current expression. The information of the bounding box is in the *Expression*[5] node, so there is no need to walk through the expression's tree to find out the expression that contains the point. When the expression is finally found, these *Expression* nodes are also useful. For instance, in the example presented in figure 4.10, if a tap is done on the last addition symbol, these are the only checks that need to be done:

- Check which node (*First* or *Last*) is nearer the point where the tap is done: the result is *Last*;

[5]It should be noted that there are two types of what is referred to, in this thesis, as *Expression* nodes. The nodes that are the root of an expression, which have kind *ROOT*, and the ones in the inner levels of the trees, which have kind *EXPR*. The two types carry the same information with exception of the *kind* value. This distinction is needed to determine the beginning of each expression.

- Check whether the point is inside the last *Expr* node: the result is false; move to the Left node;
- Check whether the point is inside the last $+$ node: the result is true and this is an operator node; thus, the node was found.

In this last example, when the last addition operator is tapped, it is mentioned that the search starts from the last *Expr* node. This happens because the algorithm starts by checking which of the first and last children of the root is nearer to the point being searched. It then proceeds with the search through the nearest node. In the case above, the node that is nearer to the point is the last one. This search for the nearest node is done to improve the efficiency of the search function. More details about the search functionality are given in section 4.7.1.

4.5.2 Creation of structured trees

To create the trees that represent mathematical expressions, the precedence and the kind of each operator involved have to be taken into consideration. To introduce the algorithm responsible for that, consider the creation of the tree for the expression $a \oplus b \ominus c \ominus d \otimes g$, where all the operators are binary, have the same precedence and their kinds are as follows:

- $\oplus$: infix;
- $\ominus$: right;
- $\otimes$: left;

The algorithm proceeds as follows. First, in the parsing process, the expression is divided in two lists, one that contains the expressions (which in this case is composed of variables only) and another that contains the operators:

```
Expressions: [a, b, c, d, g]
Operators: [⊕, ⊖, ⊖, ⊗]
```

Next, the list of operators is processed by a method called *indexesHighestPriority*. This method creates a list of the form [*Infix*, *Left* and *Right*]. *Infix* is a list that contains the indexes of all the operators that are infix. *Left and Right* is a list containing the indexes (in ascending order) of the left and right operators; this list is, in fact, a list of lists containing singleton lists for left operators, and non-singleton lists for right operators that are contiguous in the list of operators. Continuing with the example, the list returned by *indexesHighestPriority* is the following:

[[0], [[1,2],[3]]]

It should be noted that if there were operators with highest precedence, those would be processed first and, for each precedence, one list similar to the above would be created.

The next step is to process first the left and right operators (LR list). The LR list is processed from left to right but, for each list inside it, the processing occurs from right to left. This has no effect on lists of left operators since they are singleton. However, for lists of more than one right operator it has the desired effect of processing the rightmost operators first. So, processing the LR list from left to right, $[1,2]$ is processed first. Next, this list is processed from right to left, thus the operator in the index 2 is the first to be processed. Now, looking at the list of operators, that corresponds to the second $\ominus$. The operands of this operator should be retrieved, that is c and d (positions index and index+1). With this information, the first node can be created using the method *joinNodes* which returns a node, say n1. Next, the lists of expressions and operators are reorganised by replacing the operands c and d by the node n1, and removing the second $\ominus$ from the list of operators:

Expressions: [a, b, n1, g]
Operators: [$\oplus$, $\ominus$, $\otimes$]

The *indexesHighestPriority* is called again with the new lists, returning the following result:

[[0], [[1],[2]]]

Following the same procedure detailed above, the following sequence of lists is obtained (node n2 and n3 created by the method *joinNodes*):

Expressions: [a, n2, g]
Operators: [$\oplus$, $\otimes$]
[[0], [1]]

Expressions: [a, n3]
Operators: [$\oplus$]
[[0]]

At this point, no more elements are left in the LR list, so the list of infix operators should be processed. As there is only one operator, the method *createInfixNodes* is used to create the node that will connect a, $\oplus$, and n3, and to create the root node of the whole expression.

In this particular example, there is only one infix operator but, if the list of infix operators was, for instance, [1,2,4,5] (this type of list can appear when there are operators of different precedences), a list of lists would be created, where each list would contain the indexes that are contiguous (this list would be created by the method *contiguousLists*). For the list [1,2,4,5] the result would be [[1,2],[4,5]]. This list would be processed from right to left using the method *createInfixNodes*. For each list that this method receives, it creates the nodes for all the indexes in the list argument, keeping the nodes in the same level (as it should be done for infix operators). It then returns the leftmost node so that it can be connected to the remaining nodes. This list is processed from right to left since it does not matter the order in which infix operators are processed and, this way, it avoids having to recalculate the lists of indexes. If the lists were processed from left to right, the indexes would have to be updated at each step. For example, if the list [1,2] shown above was processed first, the operators at indexes 1 and 2 and their operands would be removed from the lists of expressions and operators. This would mean that the list [4,5] would no longer be consistent and would have to be recalculated.

The example above was introduced to provide the reader with an intuition of how the algorithm works. A more precise formulation of the algorithm is presented below (Algorithm 2).

Algorithm 2 createPrecedenceTree: Creates a tree from the lists *expressions* and *operators*

```
Require: $expressions$.Count>0 $\wedge$ $expressions$.Count=$operators$.Count+1;
  if $expressions$.Count = 1 then
    return $expressions$.First;
  end if
  indexes = indexesHighestPriority($operators$);
  infix = indexes.First; leftRight = indexes.Second;
  if leftRight.Count $>$ 0 then
    index = (leftRight.First).Last; bop = $operators$.index;
    left = $expressions$.index; right = $expressions$.(index+1);
    n1 = joinNodes(left,right,bop);
    $expressions$.index = n1;
    $expressions$.Remove(index+1); $operators$.Remove(index);
    return createPrecedenceTree($expressions$, $operators$);
  else
    contiguous = contiguousLists(infix);
    for i=contiguous.Count - 1; to i=0 do
      createInfixNodes(ref $expressions$, ref $operators$, contiguous.i);
    end for
    return createPrecedenceTree($expressions$,$operators$);
  end if
```

As it can be seen in Algorithm 2, if the LR list is non-empty, the size of the list of expressions decreases by one. Also, if it is empty, the method *createInfixNodes* reduces its size by at least one (but it never becomes empty). Consequently, the algorithm terminates.

The discussion above was on creating structure for expressions involving only binary operators. The above algorithm executes after the parser creates the lists of expressions and operators. On the other hand, the creation of structure for the other types of expressions does not need these lists. For example, the following code shows how the structure for an expression involving a prefix operator is created.

```
pre=DYNAMICPRE x=INT y=INT w=INT h=INT pe=primary_expression
{
    Rectangle r = new Rectangle(x,y,w,h);
    Node preNode = new Node(r,$pre.text,Node.OP_PREFIX,null,
                              null,null,$pe.local,null);
    $pe.local.Left = preNode;
    local = createExpressionNode(preNode, r);
}
```

The method *createExpressionNode* creates a node with kind *EXPR* that holds the node given as argument, and all the nodes to its right.

Note that the strategy followed by the *MST library* has some limitations. For example, when creating structure for the expression $\neg a \oplus b$, where $\neg$ is a prefix operator with lower precedence than the infix operator $\oplus$, the expression will be parsed as $(\neg a) \oplus b$, which is incorrect. For these cases, the user has to bracket the expressions. Alternatively, the library provides group and ungroup operations that can fix these inconsistencies.

Finally, the library provides limited support for combined writing of text and mathematics. So that the parser can differentiate between text and mathematics input, text has to be delimited by special characters. In the current implementation, strings are stored in a node containing a bounding box of all the strokes that compose the strings (including the delimiters). Although limited, it can be useful for short annotations in algorithms and proofs.

4.6 Spatial indexing of handwritten expressions

Many tools that provide the means to write presentations organise the presentations' content by slides or pages. So that this type of organisation could be supported by this library, the notion of *structured slides* was introduced. The idea behind the structured slides is to keep a structure that stores all the slides and their content in a way that provides easy access to their structure. For that, the library has a variable (named *structSlides*) which is an instance of the class *StructuredSlides*. This class has a single instance variable that consists of a hashtable that maps the identifiers of slides or pages to the mathematical expressions that they hold. Moreover, this structure allows the addition and deletion of expressions and can determine which expression is situated at a given point — so that the structure of an expression can be retrieved and changed when an action is triggered on it.

To retrieve the mathematical expressions contained in each slide, it is necessary to index the expressions according to their position on the slide. In other words, it is necessary to index the expressions spatially. In general, the data structure that is usually used for indexing rectangles is the *R-tree* [Gut84] structure. R-trees are tree data structures that are used for indexing multi-dimensional information; for example, the (X, Y) coordinates of geographical data [rtr12]. This data structure splits space with hierarchically nested bounding boxes (BB), that can possibly overlap. The nodes in R-trees have a variable number of entries, up to a pre-defined maximum, and each corresponds to the BB that holds its children. Nodes can be of two types: non-leaf nodes and leaf nodes. Each entry within an internal (non-leaf) node contains:

- a pointer to a child node;
- and the BB that spatially bounds all the children of the child node.

Each entry of a leaf node contains the BB of the object contained in that entry and an identifier of the object (which alternatively can be placed in the node). When nodes get full they are split. The insertion algorithms use the BB in each node to determine in which node an entry should be put. A new entry will go into the leaf node that requires the least enlargement of its BB. Similarly, the searching algorithms use the BB to check whether or not to search inside a child node, which avoids unnecessary searches inside most nodes.

Although the *MST library* assumes that the users will never write overlapped expressions, the current implementation of the structured slides is based on R-trees. More specifically, instances of the class *StructuredSlides* map the identifiers of the slides to R-trees — that is, there is a distinct R-tree for each slide. Each R-tree contains the structure of all the mathematical expressions contained within the slide.

The implementation used for R-trees is the implementation developed by Aled Morris that was ported to C# by Dror Gluska. The implementation is free software and is available at [sf12]. In this implementation, the maximum number of entries per node is 10, and when this maximum is reached, the node is divided in two, each node containing a minimum of five entries. The search algorithm, when given a rectangle, finds all the entries that contain (or intersect) that rectangle. However, as the *MST library* assumes that there is no overlap of rectangles, the implementation of the R-trees was changed so that, when it finds a node, the search stops immediately. This change makes the search more efficient, since it will avoid continuing the search through nodes that will not contain the rectangle (please note that, in the case of the *MST library*, the rectangle for the search consists of one point only).

This implementation was chosen because it was already available and its performance is acceptable. Thus, there was no need to implement another library for R-trees. Also, even though the current implementation of the *MST library* assumes that there are no overlaps, there may exist tools that need to allow overlapping. In that case, this library can still be suitable for them (with only a few changes).

It should be noted that the *MST library* is implemented so that this structure (structured slides based on R-trees) can be replaced by another structure without much work. For that, it is only needed to redefine the *StructuredSlides* class to support the new structure in a way that the current API is maintained. This consists of changing all the code that deals with R-trees in the *StructuredSlides.cs* file to deal with the new structure.

4.7 Structure editing

4.7.1 Selection and copy

When writing mathematics, content is constantly being copied and duplicated. Frequently, from one step of a calculation to the next one, only small changes are made to the expressions. Thus, one has to repeat things that were already written before when they could simply make use of a copy operation and edit solely the parts of the expression that need to change. This copy operation would save time to the user and also avoid the introduction of errors, which often happen when copying content. To be useful, this copy operation has to be simple and natural to use. The *MST library* provides features that make a copy operation straightforward and easy to use. However, to copy content effectively and to apply rules correctly, selection of content has to be accurate since the copy operation needs to know what it should duplicate. The selecting content using a lasso or a gesture circle is very easy, but, in some cases, it can fail to select the desired content. To allow accurate selection, a structured selection is available. Facilities to tap on separate elements of an expression and select content according to its structure are provided by the *MST library*. To illustrate this, consider the expression

$$a + b\times c + d + e.$$

A possible use of the selection facilities provided by *MST library* is now presented. The figures shown below are taken from a demo tool that was used to test the facilities.

A single tap on any of the symbols that form the expression selects the symbol itself. A double-tap, however, can have a different effect. If one double-taps on one of the variables (a, b, c or d) the selection obtained is still the character that was tapped. No other selection can be obtained by tapping on a variable (for variables there is no difference between a single or a double tap). However, if a double-tap is performed on an operation symbol, the selection behaves differently. In figure 4.12, the result of a double-tap on the last addition operation is shown. Because the tool knows the structure of the addition operator, it selects, together with the operation symbol, the two arguments of the operator (d and e).

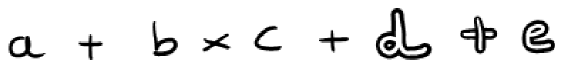


Figure 4.12: Double-tap on the last addition operator

In figures 4.13 and 4.14, the result of a double-tap on the first and on the second addition operators, respectively, returns again the addition operator together with its two arguments — but now, one of the arguments contains more than just a variable. As the multiplication has higher precedence than the addition, the tool returns it as one of the arguments of the addition. For the first addition symbol, if the tool returned only $a+b$ it would not replicate what the original expression was expressing.

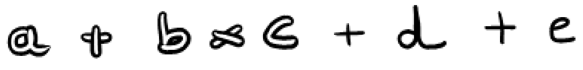


Figure 4.13: Double-tap on the first addition operator

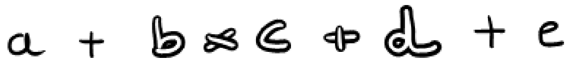


Figure 4.14: Double-tap on the second addition operator

In figure 4.15, the result of a double-tap on the multiplication operator is shown. It works the same way as for the addition operator.

Figure 4.15: Double-tap on the multiplication operator

It is possible to extend the selection by continuously tapping on one of the operators. For instance, for the expression above, if one double-taps on the first addition, $a + b{\times}c$ is selected. If, with $a + b{\times}c$ selected, one does another double-tap on the first addition, it will extend the selection to $a + b{\times}c + d$. Repeating the same process, it will finally select the whole expression — that is, it will extend the selection to contain $+e$. On the other hand, if $b{\times}c$ is selected and if one double-taps on the multiplication operator, the selection will be extended to contain the whole expression.

This structured selection is available for all the structures supported by the library (algorithms, quantifiers, etc). The selection depends on *FindNode* method to find the node that was tapped, and on the method *FullTreeSelection* to determine the selection (both methods are defined in the *Search.cs* file). The method *FindNode*, which given a point and a parent node (of kind $ROOT$ or $EXPR$) finds the node that contains the point, is a central method in the library. The selection and most structured manipulations depend on it. A high-level description of the method is presented in algorithm 3.

Basically, given a node, if its bounding box contains the point, then, if the node is an expression, continue the search through the next level of the tree; otherwise, return the current node. If it does not contain the point, check the node on the right. The implementation of the method *FindNode* contains more details than presented in Algorithm 3, since it has to deal with some special cases. For example, hints in calculational proofs are subnodes of nodes of the kind OP and that has to be taken into account in the search. Also, the method contains an optimisation that, for simplicity, is not presented in algorithm 3 but was already mentioned in section 4.5.1: it decides whether to start by the last node of the expression or by the first, according to their proximity to the point. The search starts by the node whose bounding box is nearer to the point being searched. This can improve the method's efficiency. Furthermore, for expressions that are written in just one line, if the point is between two nodes, it will return the node that is nearer to the point. This feature is useful since characters like the dot can be difficult to target with a pen, resulting in taps around its bounding box. *FindNode* should be extended, in the future, for expressions written in multiple lines.

Algorithm 3 FindNode: Finds the *node* that contains the *point* given

```
if $node \neq$ null $\wedge$ $node$.BoundingBox.Contains($point$) then
  $node$ = $node$.First;
  while $node \neq$ null do
    if $node$.BoundingBox.Contains($point$) then
      if $node$.Kind = $EXPR$ then
        return FindNode($point$,$node$);
      else
        return $node$;
      end if
    else
      $node$ = $node$.Right;
    end if
  end while
end if
return node;
```

Copying structure: Having a way to select content accurately, makes it possible to copy content accurately. For that, one needs to select what one wants to copy and then trigger the copy action. By default, the copy can be triggered by the use of the *Check* (figure 4.16) gesture.

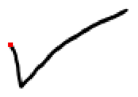


Figure 4.16: Check gesture

The point where the top-left corner of the expression's bounding box will be placed, is the one marked in red (the point of the gesture with the lowest x-coordinate). The copy operation duplicates the ink and its associated structure and changes their coordinates according to the new location. This gesture has other uses in the *MST library* and the association between the copy and this gesture can be changed by the user at anytime, as it will be detailed in sections 4.7.2 and 4.9.

4.7.2 Manipulation rules

As detailed in section 3.6, it is a functional requirement that this library provides support for algebraic manipulation of handwritten mathematical expressions. The algebraic rules that the library supports are the ones that are most often used in calculations and that consist merely of syntactic manipulation of the mathematical structures: *symmetry*, *add/remove brackets*, *group/ungroup*, *substitution of equals for equals (Leibniz)*, *distributivity*, and *factorisation*. Since mathematical expressions have a structure associated with them, it is possible to implement these rules.

The result of applying an algebraic rule is shown in the users handwriting. That is done by copying and moving the strokes around on the screen, after updating the tree that represents the expression. In figure 4.17 the result of applying symmetry to the expression $a+b$ is shown.

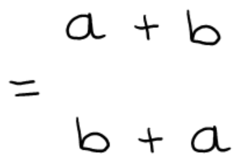


Figure 4.17: Symmetry

Figure 4.17 was obtained by, first, handwriting the expression $a+b$, recognising it and creating its structure. Next, the expression was copied below and the symmetry rule was applied. As it can be seen, the handwriting of both expressions is the same.

A high-level description of the algorithm that swaps the operands of an expression (symmetry) is shown in Algorithm 4.

Algorithm 4 Swaps the operands of the operator *node* given

```
Require: node.Kind = OP
  left = node.Left;
  right = node.Right;
  node.Left = right;
  node.Right = left;
  SwapConnections(left,right);
  UpdateBoundingBoxes(node, left, right);
```

The *SwapConnections* method interchanges all the connections of its arguments, that is, all the pointers to the left node are changed to the right node (and vice-versa) and the right node is changed so that it points to all the nodes pointed originally by the left node (and vice-versa). The *UpdateBoundingBoxes* changes the bounding box of each of the nodes to be the new bounding boxes of the strokes that they hold (since they change their positions). This method also normalises the space between the three elements involved in the operation, i.e., distance between the operator and the operands becomes the same.

It should be noted that, after the execution of this algorithm, the structure of the new expression is updated, but the ink on the screen is not. In fact, the implementation of the Algorithm 4 creates a list of *moves*[6]. These moves correspond to the changes that should be applied to the ink on the screen and, normally, they should be executed immediately after they are returned. That can be done using the method *ExecAnimations*. The same happens in all the other methods that apply algebraic rules.

For the example shown in figure 4.17, the list of moves that is created consists of three *Move* instances: one for each of the nodes involved. Each *Move* contains two instance variables:

```
private Rectangle rectangle;
private Point offSet;
```

The *rectangle* contains the bounding box that will be affected by the move, and the *offset* is the offset that will be applied to the *rectangle*. In other words, if p is the left-top corner of the *rectangle*, all the ink contained in that rectangle is moved to an identical rectangle where the left-top corner is the point $p + offset$. The class *Move* inherits its variables from the class *AnimationMoves*, which is the superclass of all the types of moves available in the library. The other types of moves and more details on the execution of the moves will be presented in section 4.8.

Separating the update of the structure from the update of the ink: In general, different systems represent ink in different ways. So, if the ink was updated inside the manipulation methods, they would have to assume a particular representation of ink. This would make it difficult (or even impossible) to use these methods within tools that treat ink differently. Separating the update of the structure from the update of the ink

[6]Algorithm 4 omits the creation of the list of moves to simplify the presentation.

means that one just needs to implement the method *ExecAnimations*, which updates the ink based on a list of moves. This method is very simple to implement as it only iterates a list of moves and applies them to the ink[7]. By default, there is an *ExecAnimations* method available which works on ink as defined by the Microsoft Tablet PC API.

Consistency between ink and structure: Since ink and structure updating are separated, programmers need to be absolutely sure that both will be updated after each manipulation or, otherwise, they have to cancel the manipulation. If care is not taken, the document can reach a point where the ink in the screen does not match the internal structure and it becomes useless. For this reason, the code that makes use of these facilities needs to assure the execution of both the structure updating and the ink adjustment.

User-defined handwritten characters: Some manipulation rules change the original expression to an expression containing more strokes than the original one. Although in some cases the new strokes can be copied from the original, there are rules that need to introduce characters that are not part of the original expression. For example, the manipulation rule that allows the addition of brackets to an expression needs to introduce two new brackets. Thus, the handwritten representation of the brackets has to be saved somewhere. Furthermore, it is desirable that this representation uses the handwriting of the user. For these reasons, the *MST library* allows the definition of user-defined characters by using the *UserDefined* class. If a programmer wants to add a new manipulation rule that needs to introduce a handwritten representation of the character 'X' to an *ink* object, they can obtain the strokes that form the character by using the following code:

```
userdefined.GetStrokes('X', ink);
```

where *userdefined* is an instance of the *UserDefined* class. In section 4.11, a more general mechanism for storing user-defined templates is presented. In the future, these two mechanisms may be unified.

[7]The method *ExecAnimations* can be more complicated than simply iterating a list of moves. In particular, to support animations (see section 4.8) the method may need to create intermediate moves.

4.8 Animation support

As mentioned in the previous section, the update of the structure is separated from the update of the ink. This is achieved by creating a list of moves while updating the structure. This list of moves can then be applied to the ink separately. The existence of a list of moves can be useful for teaching because, if moves are executed sequentially, they give the illusion of an animation. This can help the students with seeing how the rules are applied. It is important to note that the list of moves has to be carefully created so that it makes animations useful, i.e., the order in which each part of the expression is going to be moved, has to be easy to follow.

The types of moves available are:

- *Move(rectangle,offset)*: moves the ink inside the *rectangle* by an *offset*;
- *Copy(rectangle,offset)*: copies the ink inside the *rectangle* and moves the copy by an *offset*;
- *Delete(rectangle)*: deletes the ink inside the *rectangle*.

As already mentioned, these moves are executed by an *ExecAnimations* method.

In figure 4.18, some of the steps of the animation of the *Symmetry* rule are shown. It is very difficult to convey in images the dynamics of an animation; the red arrow depicted in the figure tries to give a better impression of the animation by indicating the moves.

The animation starts by moving b to its final position, then moves $+$ so that it stays equidistant from b and a, and, final, it moves a.

4.9 Gestures support

As already mentioned in section 3.8.1, the use of gestures to trigger actions is a functional requirement for this library, since they can be simple and natural to use and they usually do not interrupt the users thought. The *MST library* allows the association between gestures and actions and it provides mechanisms for editing these associations. This makes it possible for the users to adapt the use of gestures according to their needs.

The gestures supported by the *MST library* are all the *ApplicationGestures* made available through the Tablet PC API, with the exception of the *Tap* and *DoubleTap*, which

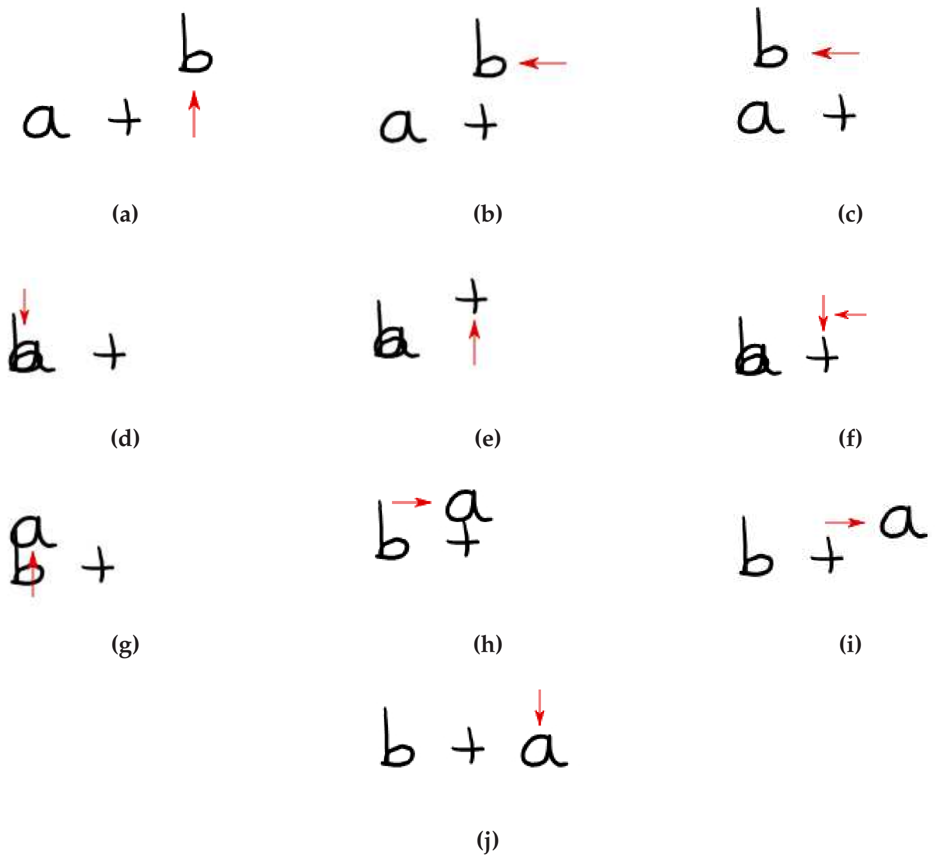


Figure 4.18: Animation for the *Symmetry* rule

are reserved for structured selection[8]. A list of the application gestures, together with notes on how to draw them, is available in [JS03, p. 356] [9].

To detect application gestures, the *MST library* relies on the mechanisms provided by the Tablet PC API. However, the way gestures are used within the library depends on the associated actions. In particular, depending on the action, the gesture points that are used may be different. For example, if one wants to associate a gesture with distributivity, one only has to consider two points of the gesture, i.e., one point for each of the operators involved. However, other manipulations may need a different number of points to be considered.

Suppose that to apply distributivity, a *SemiCircleRight* gesture starting at the operator that will be distributed and ending over the operator through which it will be distributed is used (as shown in figure 4.19). Then the first and the last points of the gesture are the only points considered. The result of applying distributivity is shown in figure 4.20.

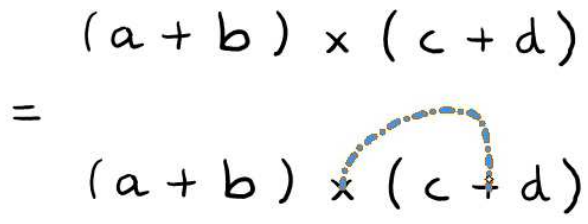


Figure 4.19: Using the *SemiCircleRight* gesture to distribute multiplication over addition

An example of a manipulation that only needs the information of one point is *Leibniz*, since it only needs to identify which expression has to be replaced. For instance, a gesture that is a good candidate for *Leibniz* is the *Check* gesture, as shown in figure 4.22. Because this gesture is simple as a trigger and *Leibniz* is used often in calculational proofs, this association seems natural. In this case, only the first point of the gesture is considered. The process of applying *Leibniz* starts with the selection of the expression

[8]If needed, programmers can easily include the *Tap* and *DoubleTap* gestures. They only need to uncomment two lines of code in the *MSTVariables* file.

[9]The list of application gestures can also be seen at
`http://msdn.microsoft.com/en-us/library/ms818591.aspx`

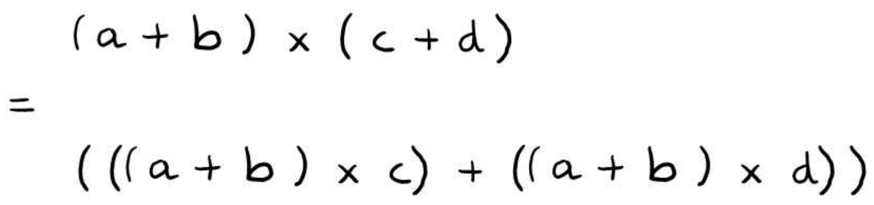


Figure 4.20: Result of applying distributivity

that is going to be used to substitute another one (as shown in figure 4.21).

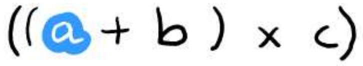


Figure 4.21: Selection of the expression to be used in *Leibniz*

Supposing that $a+b = a$, using the *Check* gesture, the expression $a+b$ is substituted by a as shown in figure 4.22. The initial point of the *Check* gesture indicates the main node of the expression being replaced. In figure 4.23 the result of applying *Leibniz* is shown.

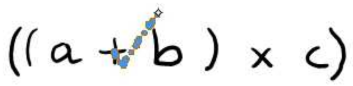


Figure 4.22: Application of *Leibniz* using the *Check* gesture

An example of a task that needs to consider several points is the deletion action. An example of a gesture that can be used for that is the *Scratchout* gesture (shown in figure 4.24). In this case, all the points of the gesture are considered since any strokes that are intersected by the gesture should be deleted.

The three examples of associations between gestures and actions shown above are provided by default with the library. These gestures were defined to be intuitive or simple enough for the particular tasks. Besides using gestures for manipulation rules and editing tasks, gestures are also used within the library to open editing windows (the interface shown in section 4.11.1 is an example).

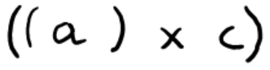


Figure 4.23: Result of applying *Leibniz*

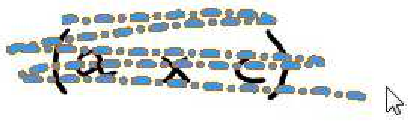


Figure 4.24: The *Scratchout* gesture

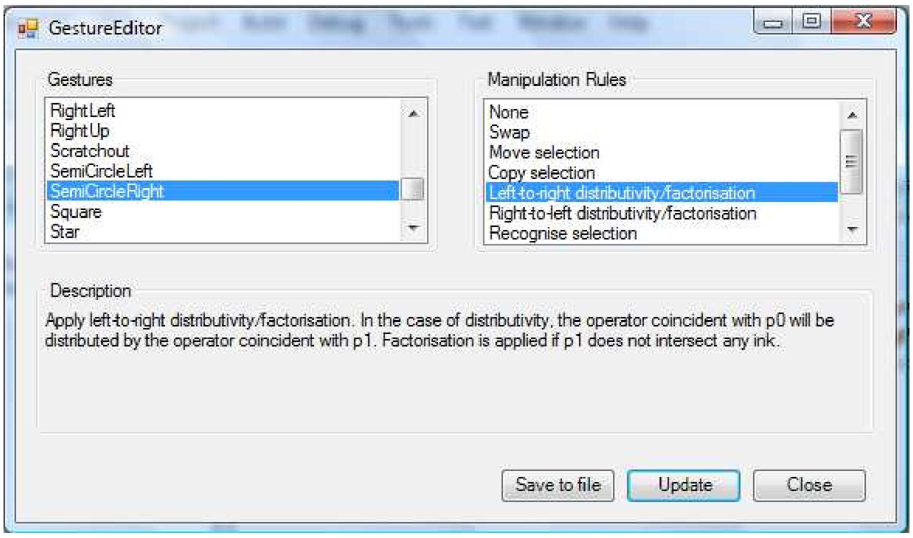
Adding new actions to be triggered by gestures: All the gestures facilities are available in the module *Gestures*. The associations between gestures and actions that are defined are held by a global variable of the library named *MSTGestureManips*. To add a new gesture to the system, a new instance of the class *MSTGestureManip* has to be created. In algorithm 5, the pseudocode of the delete action and how to start its execution is shown (where *g* is the object representing a gesture):

Algorithm 5

```
if MSTGestureManips.Contains($g$) then
  if MSTGestureManips[$g$] = MANIP_DEL then
    find the node $n$ that contains the first point of gesture $g$;
    if $n \neq$ null then
      delete structure from slide;
    end if
    delete all the strokes that intersect the points of the gesture $g$;
  end if
end if
```

To associate gestures with actions, an editor is provided. Figure 4.25 shows the editor as it is shown to the users.

If a new action is added to the library, then it should be made available through this editor so that it can be associated with gestures. To do that, one only has to add the identifier of the action to the global variable *MRulesNames*. It should be noted that one action can be associated with several gestures. However, gestures can only be



Apply left-to-right distributivity/factorisation. In the case of distributivity, the operator coincident with p0 will be distributed by the operator coincident with p1. Factorisation is applied if p1 does not intersect any ink.

Figure 4.25: Gestures' Editor

associated with one action (otherwise, it would be difficult, if not impossible in some cases, to identify which action should be started). Note, however, that it is possible to define actions that perform differently depending on the context. For example, before, we have used the *Check* gesture to both copy structure (section 4.7.1) and to perform *Leibniz*. This was possible because both are defined within the same action. For that action, a test determines if there is a structured expression under the initial point of the *Check* gesture: if there is, then replace that expression, otherwise, just copy to that location the expression that was selected.

4.10 Markup and binary outputs for mathematical expressions

As already mentioned in section 4.6, the library contains a *StructuredSlides* class that associates each slide/page of the presentation, with an *R-Tree* that contains all the expressions within that slide. All the objects involved in this structure are serializable[10]. Thus, they can be written to a file and loaded using the standard mechanisms provided by C#, or by using the methods *SaveMSTStructure* and *LoadMSTStructure* provided by the *MST library*. It should be noted that the library does not provide any methods to save ink, since the applications that use this library should already provide facilities to save the content of their documents (which includes the ink). To save and load the

[10]The original implementation of the *R-Trees* was not serializable. Thus a small modification had to be done to the original code.

structure at the same time as the content of the documents, programmers only have to use (within their implementation) the methods mentioned above. Given a *FileInfo f*, these methods will save (or load) the file *(f.name).mst* and this can be combined with the custom save (or load) of the applications.

The *MST library* also provides methods to save and load single expressions together with the ink in a plain text format. The class *XML* contains these methods. The text format created has the following structure:

```
<ink> InkML that represents the ink </ink>
<tree> TreeML that represents the structure of the document </tree>
```

Basically, the file is an XML file that describes the content of the document. The InkML [ink11] format is an XML data format for representing digital ink data that is input with an electronic pen or stylus as part of a multimodal system. The *TreeML* was defined for this project with the purpose of saving the structure of expressions. It consists only of a few XML tags that describe the structure of the rose trees. The TreeML tags that are currently defined are the following:

- `<node> ··· </node>` Describes a node;
- `<bb> ··· </bb>` Holds the information of a bounding box (x, y, width and height);
- `<reco> ··· </reco>` Character that represents the recognition of the handwritten held by the node;
- `<kind> ··· </kind>` Kind of the node (1 to represent an expression, 2 to represent an operator, etc);
- `<children> ··· </children>` Contains the nodes that are children of the node being described.

The InkML is written to the file using the *InkML toolkit* (InkMLTk [ink08]) which aims to provide a suite of tools for working with InkML documents. In particular, the converter *ISF2inkML* that is provided by the InkMLTk is used to convert an Ink object into InkML. The result of using this converter forms the first part of the file. The second part (the TreeML description) is generated by code created for this library, and basically walks through the tree and writes to the file the representation of this tree.

Reading these files is straightforward: the InkML is loaded using the InkMLTk and the TreeML is parsed by the library to create the rose tree that the TreeML represents. To read the InkML, the converter *InkML2ISF* is used. This converter transforms the InkML into an ISF from which an Ink object can be immediately extracted and shown on the screen. However, there is a problem with this converter, which results in the loss of some information about the strokes. Consequently, strokes are not loaded exactly as they were in the original document (they appear to be thinner than originally). This problem has already been reported to the InkMLTk community.

A fragment of the plain text format described above, showing only the representation of a node, is presented below:

```
<ink>
 ...
</ink>
<tree>
 ...
<node>
<bb>4377 2439 1078 542</bb>
<reco>a</reco>
<kind>3</kind>
</node>
 ...
</tree>
```

Methods to save and load *StructuredSlides* to and from the plain text format are not provided in the library, because *StructuredSlides* associate arbitrary *objects* representing slides with *R-trees*. Different systems have different definitions for slides. For example, slides may contain information about the background image or colour of the slide while others may have a timestamp. This makes it impossible to predict which tags should be used to represent the information. However, this mechanism for creating an XML description of an expression is made available so that programmers can combine it with their own file definitions if needed.

4.11 Experimental features

As already mentioned, the aim of this project is to provide a flexible and straightforward way to teach mathematics. For that reason, some experimental features[11] were developed. In the next two sections, these features are presented.

4.11.1 Handwritten templates

A major handicap for this project is the recognition of mathematics. As it is difficult to recognise expressions, it becomes difficult to create structure for them. For this reason, the *MST library* contains an experimental mechanism for handwritten templates. Users are able to store expressions with structure and use them whenever they want. These expressions can be seen as templates and, using substitution of equals for equals, can be used to write more complex expressions. For example, the user may define templates for the *GCL* language that can be reused later, avoiding the burden of having to recognise it again.

In the current system, if the user wants to use a predefined template, they can open a template selector window using a gesture (currently the *Square* gesture[12]). If a template is to be inserted, it is inserted at the first point of the gesture. If there is any expression under the point, *Leibniz* is applied (this is a way of writing complex expressions). Figure 4.26 shows the current version of the template selector. The templates shown are a subset of *GCL* expressions that can be composed to write complete algorithms. Categories can be used to organise the templates.

The idea of these templates was inspired by the Math∫pad stencils. However, compared to Math∫pad, the *MST library*'s implementation is very primitive. For example, in Math∫pad, when a stencil is updated, the document is also updated to reflect the stencil's changes. More importantly, in Math∫pad, one is able to create stencils representing new structures. None of these features is currently available in the *MST library*.

[11]These features are called experimental because they were developed to test concepts that were not developed in depth.

[12]Please recall that the user can change this gesture at anytime.

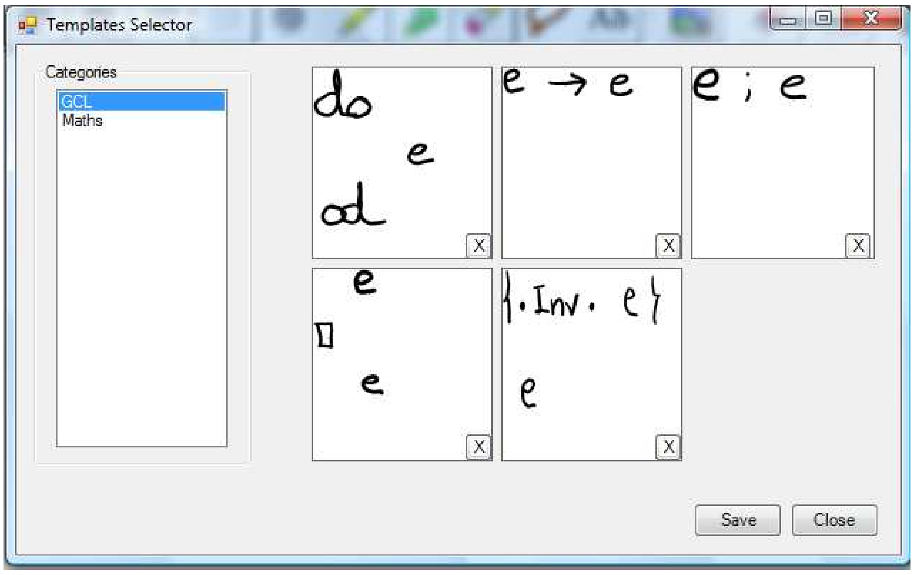


Figure 4.26: Templates selector

4.11.2 Automatic space adjustment

One feature that is implemented but is not included in the *MST library* is the automatic space adjustment of mathematical expressions. This comes as a suggestion of a feature that can improve the users experience while writing mathematics. The idea behind this feature is that the user does not have to stop to adjust the spacing to continue with their writing when no space is left: the tool can do that for them. For example, if the user is editing the summation quantifier shown in figure 4.27,

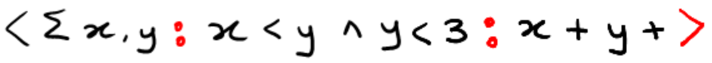


Figure 4.27: Summation

the characters marked in red are adjusted automatically as the user writes inside the quantifier expression. If the user is writing near one of these characters, the expression extends automatically, relieving the user from the work of having to adjust it manually and having to interrupt what they are doing. In the example of figure 4.27 the term's expression is not finished, so if one starts to write the rest of the expression, the '⟩' symbol will move to the right, allowing space for finishing the term.

This feature is not included in the library since it depends on how ink is collected and the author does not know any good way of making it available in a general form.

CHAPTER 5

Integration into external tools

In this chapter, a general description of how to extend tools using the *MST library* is given. To illustrate that the library can be easily integrated, this chapter also describes the library's integration into the *Classroom Presenter*.

5.1 Introduction

Creating a tool for the particular needs of a group of users may make it unsuitable for another group. A particular worrying aspect of this project was that, if an application was provided instead of a class library, it would be too oriented to algorithmic problem solving, which could lead it to quickly become outdated and unused. For this reason (and others mentioned in section 3), the main result of this project is a class library that can be easily integrated into other tools. This makes it possible for developers to freely use within their tools features of the MST library that they might find useful, and in the way they find best suited. Furthermore, programmers can choose features according to what makes sense to the users of their applications. For example, in tools for document preparation only, automatic manipulation will be useful while animations will not make sense — since they exist for educational purposes and, in principle, a user of a document preparation tool already understands how the rules are used.

The goal of this chapter is to show that integrating the *MST library* into other applications is straightforward, especially for *.NET* applications. In section 5.2 a brief description of how tools can make use of the library is given, and in section 5.3 it is described how the library was used to extend Classroom Presenter, a system developed to create interactive presentations using a Tablet PC, into a structure editor suited for teaching algorithmic problem solving.

5.2 How to integrate the library

To use the *MST library* within a tool one has to import the library into a project, extend the user interface to allow the use of the imported features, and start using them. In some cases, small adaptations have to be made to match the types used by the tool and the ones that the methods of the library are expecting. However, the library was created to be as general as possible in order to avoid this type of problem.

The *MST library* is divided in modules. Each module supports one of the main features of the library. The reason for this is that it makes the functionalities of each module independent from other modules. [1]. So, if a programmer wants to use, for example, the gestures' facilities provided by the library, they only have to use the functions exported by the *Gestures* module.

The modules provided by the library are the following:

- Recognisers: for connecting recognisers and for recognition of the input;
- Parsers: for connecting parsers and for creating the structure representing the syntax of the input;
- Structure: for operations on the structure (like structured selection and animations);
- Gestures: responsible for associating gestures with actions.

In the next four subsections, an overview of each module is given.

5.2.1 Module *Recognisers*

The module *Recognisers* is responsible for all the recognition facilities and for the connection to available recognisers. To use the facilities provided by this module one has to import the *MST.Recognisers* namespace and, if one does not want to use the default recogniser (the module has already one recogniser defined), one has to define

[1]However, there are internal dependencies — for example, the module *Gestures* assumes the structure defined in the module *Structure*.

a *MSTRecogniser*, i.e. define a class that implements the interface *MSTRecogniser*. This interface has one single method with signature:

    public RecAlternatives mstRecognize(Strokes s);

which has to be defined by the classes that implement the *MSTRecogniser*. This *mstRecognize* method is the central part of the recognition. The aim of this method is to recognise the strokes given as input and return a *RecAlternatives*, which is a list of pairs of the form (*Recognition, BoundingBox*), i.e. the recognition of a given character together with its bounding box. This list of pairs consists of the results returned by the recogniser being used. These can then be shown to the user so that they can choose the correct recognition result.

This process can be done step-by-step by the programmer, but to help with the process of calling the recogniser and showing the recognition results to the user, the *CallRecogniser* class was created. This class has four instance variables:

1. *Strokes stroke*: strokes to be recognised;
2. *MSTRecogniser rec*: recogniser to be used;
3. *String BoundingBoxes*: bounding boxes of the strokes being recognised;
4. *String recResultSelected*: recognition result;

and it can be used in two ways:

1. *CallRecogniser(Strokes strokes)*: creates an instance of the class with the strokes given as argument and with the default recogniser — which is defined to be the recogniser from Maple;
2. *CallRecogniser(MSTRecogniser recognizer, Strokes strokes)*: creates an instance with the strokes given as argument and the recogniser *recognizer* — the argument of the method.

After the creation of an instance of the class *CallRecogniser*, the method with signature:

    public String recognize();

can be called on the instance. This method will recognise the instance's strokes using the instance's recogniser. By the end of its execution, the values of *BoundingBoxes* and *recResultSelected* are set and can be accessed. During the execution of this method a window with the recognition alternatives is shown to the user. The user is given the freedom to choose the correct recognition or even edit the recognition results in case they do not match the input. The value selected/introduced by the user is the one that will be set as the *recResultSelected*.

Using the facilities provided by the module *Recognisers*, programmers do not have to worry about all the intermediate steps between sending strokes to the recogniser and obtaining a recognition result. The only work that has to be done by the programmer is to create a *MSTRecogniser*, in case they want to use a specific recogniser. Otherwise, the default recogniser is used and the programmer does not even have to worry about that.

In summary, to make use of the module *Recognisers* the following steps are needed:

1. import *MST.Recognisers*;
2. define class that implements the *MSTRecogniser* (optional);
3. create an instance of the class *CallRecogniser*;
4. and, finally, call the method *recognize* on the *CallRecogniser*'s instance.

5.2.2 Module *Parsers*

The module *Parsers* allows the connection of parsers, and is responsible for the creation of structure for handwritten input.

Similarly to the module *Recognisers*, to use the facilities provided by the module *Parsers*, one has to import the *MST.Parsers* namespace and define a *MSTParser* — a class that implements the interface *MSTParser*. This interface also has one single method with signature:

```
public Node mstCreateStructure(String reco);
```

which has to be implemented by the classes that implements the *MSTParser*. The *Node* returned by the method above corresponds to the root of the tree that holds the whole structure that represents the string given as argument. As mentioned in chapter 4, the string argument is composed of the recognition string and the information of the bounding boxes of the strokes that where recognised. An example of such a string is:

"4366 3168 3518 949 a 4366 3541 1002 576 + 6031 3496 689 474 b 7323 3168 561 835".

This string represents the expression $a + b$ where each set of four integers forms a bounding box of the form (x coordinate, y coordinate, width, height), being the first set of four integers the bounding box of the whole expression and each set of four values after each character the character's bounding box. Thus, in order to make the *mstCreateStructure* method work, a string of this form has to be created. To avoid the work of having to do each step of the parsing process separately (like having to create the string that the *mstCreateStructure* is expecting as input) a class called *CallParser* was created. This class has three instance variables:

1. *String exp_reco*: string to be parsed;
2. *String boundingBoxes*: string representing the bounding boxes of the symbols that form the expression being parsed;
3. *MSTParser parser*: parser that will be used;

and similarly to the *CallRecogniser* described in the previous section, this class can be used in two ways:

1. *CallParser(String exp_reco, String boundingBoxes)*: creates an instance of the class with the string *exp_reco*, the bounding boxes *boundingBoxes* and with the default parser. The default parser is defined to be the *DynamicAntlr* parser that was described in chapter 4;
2. *CallParser(MSTParser parser, String exp_reco, String boundingBoxes)*: same as method above but, in this case, the parser is set to be the *parser* given as input.

Note that the programmer does not have to calculate the bounding boxes of the characters, as these can be created by the module *Recognizers*.
After the creation of an instance of *CallParser*, the method with signature:

```
public Node createStructure();
```

can be called on the instance. This method will create the string that combines the recognition result with the bounding boxes, send it to the parser and return the root node of the tree that represents the expression.

In summary, to make use of the module *Parsers* the following steps are needed:

1. import *MST.Parsers*;
2. define class that implements the *MSTParser* (optional);
3. create an instance of the class *CallParser*;
4. and call the method *createStructure* on the *CallParser*'s instance.

5.2.3 Module *Structure*

The module *Structure* is the module responsible for all the structure definitions and facilities. In this section, we concentrate on the use of its facilities. For a more technical description of this module, please see chapter 4.

To make use of this module, one has to import the namespace *MST.Structure*. Structured selection, manipulation rules and animations are all facilities provided by this module. Of course that these were created for the type of trees that were defined in this library and, as such, to make use of these the structure to which they are applied has to be consistent with the structure defined in this project.

The *Search* class is the one responsible for structured selection. Its main methods are *FindNode* and *FullTreeSelection* with the following signatures:

```
public static Node FindNode(Point p, Node n);

public static Strokes FullTreeSelection(ArrayList SelectedNodes,
                                         Node n, Point p, Strokes wholeScreen);
```

Given an expression represented by the node n and a point p, the method *FindNode* finds the (sub-)node that contains the point p. For example, if the arguments are the root node of a tree that holds the expression $a + b \times c$, and a point intersecting the strokes of the multiplication sign, *FindNode* returns the node that hold the multiplication sign ($\times$). This method is often used to identify the node where the user has touched with the pen.

The method *FullTreeSelection* is responsible for the structured selection of expressions. The argument n corresponds to the root of the tree that holds the slide where the method was triggered, *SelectedNodes* is the list that holds all the nodes that are selected and to which new nodes will be added, *p* is the point where the structured selection was triggered, and *wholeScreen* holds all the strokes in the context. This method returns a *Strokes* object containing all the strokes that should be put in the selection.

The class *ManipulationRules* holds all the methods related with the manipulation rules and animations. The rules currently defined are **reverse**, **group**, **ungroup**, **add brackets**, **remove brackets**, **distributivity**, **factorisation**, and **Leibniz**. The signatures of the methods available are the following:

1. *public static ArrayList reverse(Boolean animation, Node root, Strokes selected, Ink ink, Form f)*: reverses the strokes *selected* according to their mathematical structure;
2. *public static Node group(Node root, Strokes selected)*: groups the strokes *selected* according to their mathematical structure;
3. *public static void ungroup(Node root, Strokes selected)*: ungroups the strokes *selected* according to their mathematical structure;
4. *public static void addBrackets(Node root, Strokes selected, UserDefined currentUser, Ink ink)*: adds brackets to the strokes *selected*. It needs to access a *UserDefined* instance so that it can retrieve the user's handwritten representation of the brackets;
5. *public static void removeBrackets(Node root, Strokes selected, Ink ink)*: removes the outermost brackets of the strokes *selected*;
6. *public static ArrayList distribute(Boolean animation, Ink ink, Node initial, Node final)*: distributes the operator in the node *initial* over the operator in the node *final*;
7. *public static ArrayList factorise(Boolean animation, Strokes selected, Node root, Node initial)*: factorises the operator in the node *initial* and its left operand out of the whole expression represented by the strokes *selected*;
8. *public static Node Leibniz(Node n, Node n0, Point inkPoint, Ink ink, Strokes selected)*: replaces the expression whose main node is *n* by the expression whose main node is *n0*. The *inkPoint* is the point where the trigger for the rule was detected (for instance, a tap), and *selected* are the strokes that will replace node *n*'s strokes (their representation has root node *n0*).

In the above, the argument *animation* indicates whether the operation should be animated, the *root* argument is the root of the tree that represents the expression to which the rule is being applied, the argument *ink* is the *Ink* object that contains the whole ink of the slide, and the form *f* is the form where the action is being triggered. For all these methods, actions are executed only if they maintain a structure that makes sense in mathematical terms (for example, they will not replace "$a + b$" by "$a +$" since addition is a binary operator).

As previously mentioned, the module *Structure* has all the definitions for the structures used within the methods of the *MST library*. As detailed in section 5.2.2, the structure for a single mathematical expression can be obtained by using the *createStructure()* method. If the programmer wants to store multiple expressions within the same spatial context — for example, multiple expressions in a single slide — the class *StructureSlide* can be used[2]. This class has one single instance variable (*Hashtable structSlides*) which associates identifiers of slides with the *RTree* that holds all the expressions present in that slide. To use this class one just needs to create an instance (*new structuredSlide()*) and use the methods *add*, *get*, and *del* to add, access and delete expressions stored in the *RTrees*. Some of the methods described in this chapter assume the existence of a slide identifier (*slideId*) and using this class can simplify the use of those methods. Nonetheless, existent structures can still be used either by mapping them into a *StructuredSlide* or by defining a new variable in the *MSTVariables* class with the name *structSlides* that provides methods with the same signature as the ones available in the *StructuredSlide*'s class.

5.2.4 Module *Gestures*

The module *Gestures* is responsible for the association of gestures with actions. As already described in chapter 4, users can edit, in runtime, the association between gestures and actions making it possible for each user to choose whichever gestures they find more convenient. To make use of this module the namespace *MST.Gestures* needs to be imported. The *Main* class in this namespace contains the method with the following signature:

```
    public static bool exec_gesture(Object slideId, Object selectionSlideId,
            ApplicationGesture g, ArrayList pts, Ink ink, Strokes selection, Form f1);
```

With this method, given the slide where the action should be done (identified by *slideId*) and the *selectionSlideId* which is the id of a slide with a selection (it is needed for actions

[2]As explained in chapter 4, this class is based on range trees.

that require a set of selected strokes — for instance, Leibniz needs an indication of which strokes will replace the ones that will be removed), given an application gesture *g*, a list of points (*pts*) that form the stroke that triggered the gesture, the *ink* of the current slide, the selected strokes (*selection*) and the form where the action was triggered (*f1*), it executes the action associated with the given application gesture. The form needs to be given as an argument so that its screen can be updated after the moves of its strokes. The information about which action should be executed is retrieved from the hashtable *MSTGestureManips*, which is defined in the class *MSTVariables*. The *MSTGestureManips* associates gestures with instances of the class *MSTGestureManip*. This class has the following instance variables:

1. *String name*: name of the manipulation;
2. *String description*: description of what the manipulation does.

The available *MSTGestureManip* instances are defined in the class *MSTVariables*. If other instances are needed, they should be added to this class. By default, some gestures are already associated with actions. To change the default associations and add new associations, the form *GestureEditor* can be used.

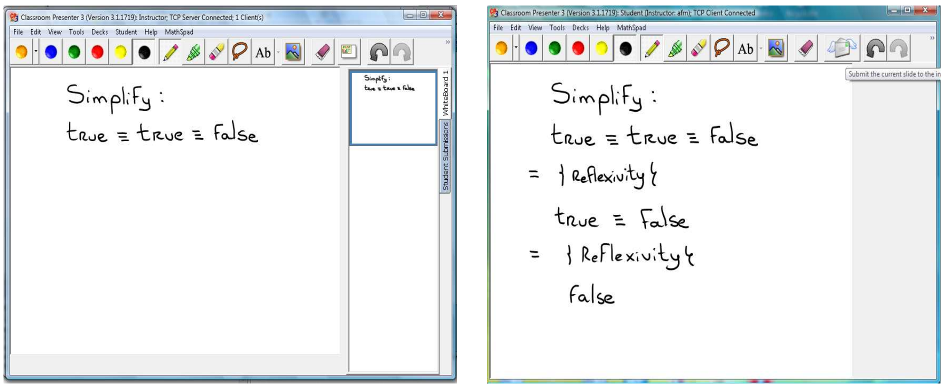
5.3 An Example: Extending Classroom Presenter

Classroom Presenter (CP) [AAHW04, AAS$^+$] is a tool which aims to facilitate the active learning in the classroom by the use of networked Tablet PCs and digital ink. The lecturer can create a network session to which the students can connect using their own Tablet PCs. Through that network, the lecturer can distribute electronic slides, which will be received by the students' computers. The lecturer can annotate the slides and the annotations are sent to the students and to a public display in real-time. Thus students can follow through their own computers what the lecturer is doing and they can add their own comments to presentations, which they can then submit through the network to the lecturer. The lecturer can also pose activities for the students to solve in their own machines and send their solutions back to him. The lecturer receives the students' submissions in a private filmstrip and can show them to the whole class if desired.

As *CP* is an application that is aimed for teaching and emphasises interactivity in the classroom, it is a good example of an application that can make use of the functionalities that the *MST library* provides. Moreover, as this tool has already many functionalities that are useful for teaching (like presentations being shown in real-time in the students' computers, inking with different colourings and powerpoint slides management), it can be used as a basis for the construction of a standalone, ready-to-use structure editor of handwritten mathematics.

Extending the *CP* has posed many difficulties, mostly because of the non-existent documentation. However, the main features described in previous chapters were successfully integrated into the *CP*. This supports the claim that the *MST* library is usable and general enough to be used by tools that were created with no attempt to make them compatible with the library.

Before describing how the integration was done, an example scenario of the use of the *CP* is presented in the figures 5.1 and 5.2 below.


Classroom Presenter 3 (Version 3.1.1719): Instructor; TCP Server Connected; 1 Client(s)
File Edit View Tools Decks Student Help MathSpad
Ab
Student Submissions
WhiteBoard 1
Classroom Presenter 3 (Version 3.1.1719): Student (Instructor: afm); TCP Client Connected
File Edit View Tools Decks Help MathSpad
Submit the current slide to the in

Simplify :

true ≡ true ≡ false

Simplify :

true ≡ true ≡ false

= { Reflexivity }

true ≡ false

= { Reflexivity }

false

(a) Teacher's screen: teacher poses a problem

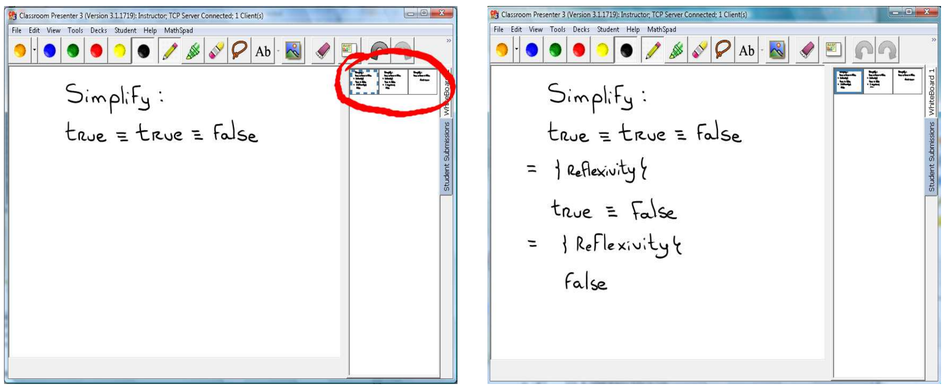
(b) Student's screen: student writes a solution

Figure 5.1: Classroom Presenter in use

In figure *5.1a*, the lecturer writes a problem for the students. In figure *5.1b*, one student's screen is shown with the answer that the student is going to send to the lecturer.

Figure *5.2a* shows the lecturer's screen — the answers of the students are on the top-right corner of the screen (highlighted in red). Three answers were received. In figure *5.2b*, the lecturer's screen is once again shown but, this time, the lecturer has selected one of the students' answers to be shown to the audience.

The interaction in the classroom can be improved using this tool (tests performed confirm this; see section 6.3). Most students do not like to raise their hands and provide an answer during a lecture. Writing it, sending it back to the teacher and, for example, having it shown without being identified, can make them engage better with the mate-



Simplify:

$true \equiv true \equiv false$

Simplify:

$true \equiv true \equiv false$

$= \{ \text{Reflexivity} \}$

$true \equiv false$

$= \{ \text{Reflexivity} \}$

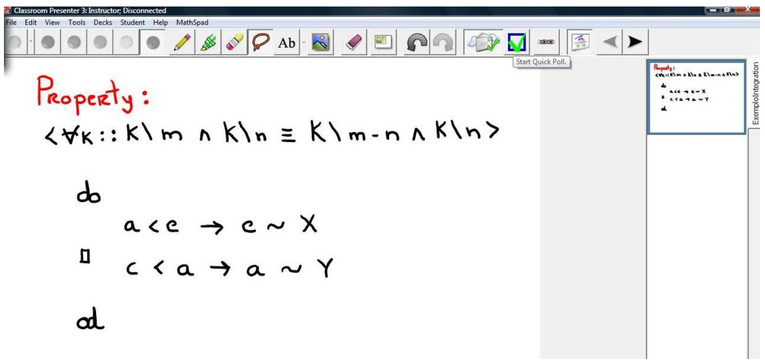
$false$

(a) Teacher receives solutions from students **(b)** Teacher selects one solution

Figure 5.2: Classroom Presenter in use

rial. This also gives the teacher the opportunity to add as much or as little content to its presentations during the lecture, being assured that the students will have a copy of it immediately and that they can add their own comments to it while the content is being taught.

The extended version of the *CP* now supports all the main features provided by the *MST library*. The following example shows some of those features in use. Figure 5.3 shows a slide with an intermediate step of a derivation of Euclid's algorithm:



Property:

$\langle \forall k :: k \backslash m \wedge k \backslash n \equiv k \backslash m{-}n \wedge k \backslash n \rangle$

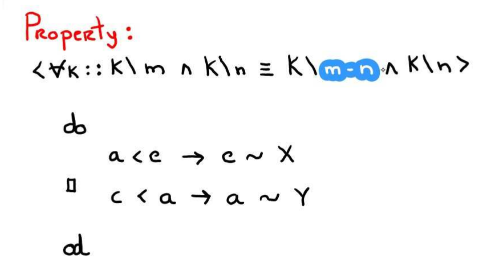
```
do
     a < c → c ~ x
□    c < a → a ~ y
od
```

Figure 5.3: Intermediate step of a derivation of Euclid's algorithm

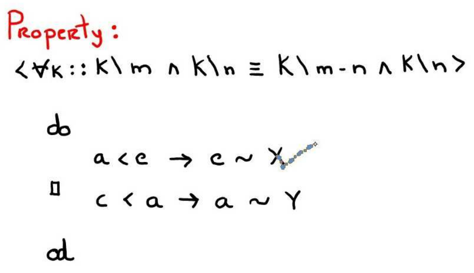
As structure is created for all the mathematical expressions present in the screen (including the algorithm), the structure can be manipulated to continue with the derivation. Note that $\sim$ in the example represents the assignment to a variable ($c \sim x$ assigns the value of x to c).

In figure 5.4a the expression $m - n$ is selected in order to substitute x by it; in figure 5.4b

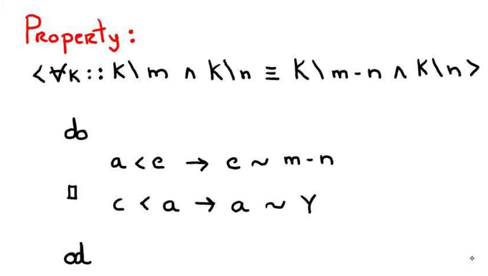
Leibniz is used to make the substitution — the tick gesture triggers Leibniz; in figure 5.4c the result of the substitution is shown.



(a) Select expression



(b) Use tick gesture to trigger Leibniz



(c) Result

Figure 5.4: Derivation of Euclid's algorithm (Part I)

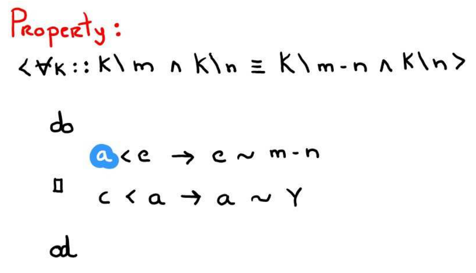
In figure 5.5, the subexpression n in $m-n$ is replaced by a using Leibniz.

In figure 5.6, a step similar to the previous one is done: m in $m-a$ is replaced by c.

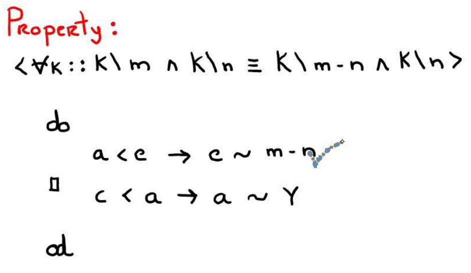
In figure 5.7, y is replaced by $c-a$ using, again, Leibniz.

Finally, in figure 5.8, $c-a$ is selected, the manipulation rule *reverse* is triggered through the menu and the final result is obtained.

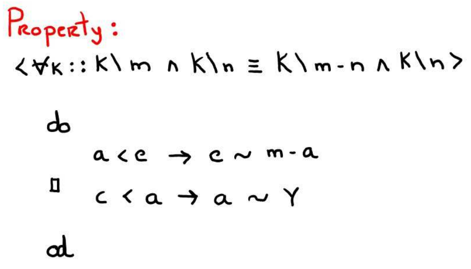
This example is simple and most of the steps could have been done by simply deleting and copying strokes. However, one should not forget that this is only an illustration of what can be done and these features scale to more complicated problems, where much larger expressions are involved. The features provided by the *MST library* are a simple and reliable way of doing this kind of calculation, since they assist with the manipulations involved and help to avoid the introduction of error — which happens often when using copy-and-paste.



(a) Select expression

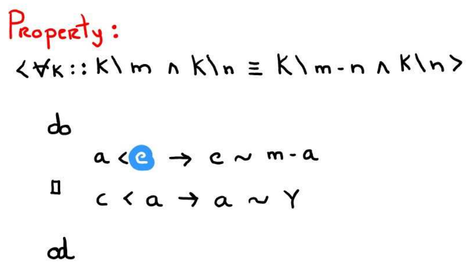


(b) Use tick gesture to trigger Leibniz

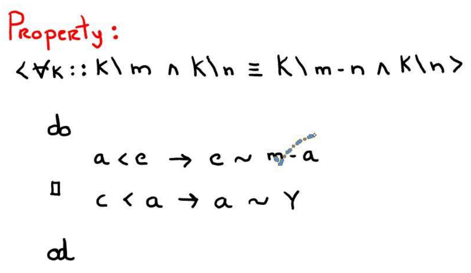


(c) Result

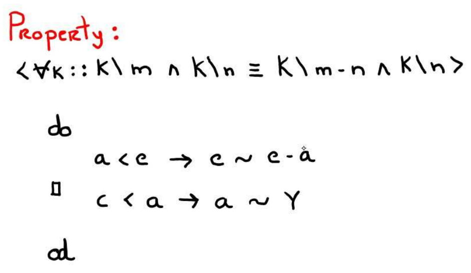
Figure 5.5: Derivation of Euclid's algorithm (Part II)



(a) Select expression

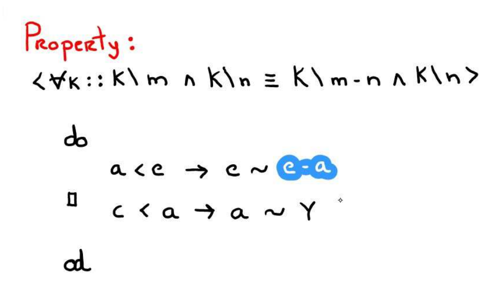


(b) Use tick gesture to trigger Leibniz

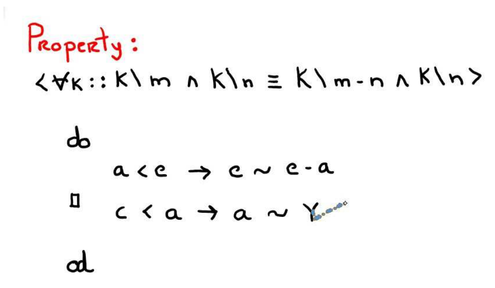


(c) Result

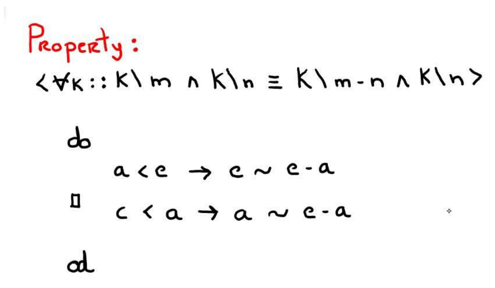
Figure 5.6: Derivation of Euclid's algorithm (Part III)



(a) Select expression

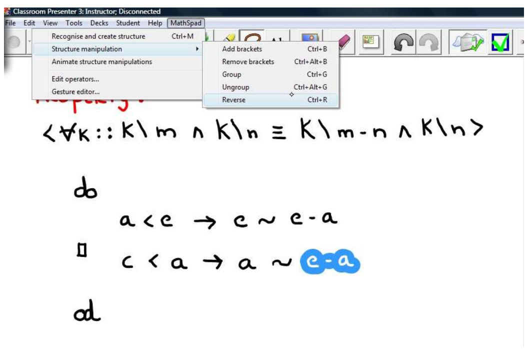


(b) Use tick gesture to trigger Leibniz

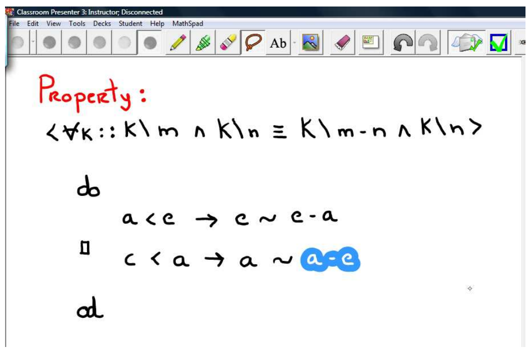


(c) Result

Figure 5.7: Derivation of Euclid's algorithm (Part IV)


Classroom Presenter 3: Instructor; Disconnected
File Edit View Tools Decks Student Help MathSpad
Recognise and create structure Ctrl+M
Structure manipulation
Animate structure manipulations
Edit operators...
Gesture editor...
Add brackets Ctrl+B
Remove brackets Ctrl+Alt+B
Group Ctrl+G
Ungroup Ctrl+Alt+G
Reverse Ctrl+R
⟨∀k :: k\m ∧ k\n ≡ k\m-n ∧ k\n⟩
do
a < c → c ~ c-a
▯ c < a → a ~ c-a
od

(a) Select expression and use menu to trigger *reverse*


Classroom Presenter 3: Instructor; Disconnected
File Edit View Tools Decks Student Help MathSpad
Property:
⟨∀k :: k\m ∧ k\n ≡ k\m-n ∧ k\n⟩
do
a < c → c ~ c-a
▯ c < a → a ~ a-c
od

(b) Result

Figure 5.8: Derivation of Euclid's algorithm (Part V)

5.3.1 Technical details

The version of the *Classroom Presenter* that is used in this project is *3.1 Version 1719*. *CP* is written using the *Microsoft's Tablet PC API* and, in particular, it uses the *RealTimeStylus* class which provides lower-level, higher performance access to stylus input than standard ink collection. The *RealTimeStylus* is designed to provide real-time access to the data stream from the tablet pen. Since *MST library* was initially created to use with a standard ink collection (in particular, the use of an InkOverlay), small adaptations had to be done. For instance, the method *ExecAnimations* of the *MST library*:

```
public static void ExecAnimations(ArrayList moves, InkOverlay inkOver)
{
   ExecAnimations(moves, inkOver, null);
}
```

assumes the existence of an InkOverlay, which does not exist in the case of the *CP*. But to solve this type of incompatibilities, methods as simple as the following:

```
public static void ExecAnimations(ArrayList moves, Ink ink, Form f)
{
   InkOverlay io = new InkOverlay();
   io.Ink = ink;
   ExecAnimations(moves, io, f);
}
```

were created. The few type incompatibilities that were found were solved with the addition of simple wrapper methods like the one shown above[3].

One of the goals of the integration of the *MST library* into the the *CP* was to keep the changes to the *CP* as minimal as possible. This goal was achieved since the integration of the library involved adding code instead of changing the original source code. To achieve the goal of adding the library's features to the *CP* two main additions were made to the CP's implementation:

- inclusion of the *LassoPlugin*;
- and the creation of the menu *MathSpadMenu*.

[3]The methods that were created to overcome the incompatibilities with the *Classroom Presenter* are now part of the library and make the library suitable for tools that use the RealTimeStylus.

LassoPlugin This class is responsible for the lasso selection tool. The *LassoPlugin* class was already written but, for some unknown reason, it was not being used within the *CP*. Now it is used and, with some additional code, it is now responsible for all the selection facilities and for the triggering of gestures. To enable gesture recognition in the *CP* the following code was added to the *PresentationLayout.cs* file:

```
GestureRecognizer myGestureRec = new GestureRecognizer();
ApplicationGesture[] gs = { ApplicationGesture.AllGestures };
myGestureRec.EnableGestures(gs);
myGestureRec.Enabled = true;
this.m_RealTimeStylus.AsyncPluginCollection.Add(myGestureRec);
```

Here, a *GestureRecognizer* is created and enabled, and the interest of the *myGestureRec* in all possible gestures is declared. The *myGestureRec* is then added to the *AsyncPluginCollection* so that it can interact with the data stream from the tablet pen. To indicate what to do when a gesture is recognised, the following code was added to the *LassoPlugin*:

```
void IStylusSyncPlugin.CustomStylusDataAdded(RealTimeStylus sender,
                                                    CustomStylusData data)
{
   (...)
   if (data.CustomDataId == GestureRecognizer.GestureRecognitionDataGuid)
   {
      GestureRecognitionData grd = (GestureRecognitionData)data.Data;
      GestureAlternate ga = grd[0];

      if (afmCollected.Count != 0)//If the lasso has collected data.
      {
         using (Synchronizer.Lock(this.m_Display.SyncRoot))
         {
            using (Synchronizer.Lock(this.m_Sheet.SyncRoot))
            {
               bool g = MST.Gestures.Main.exec_gesture(
                              this.m_Display.Slide.Id, afmSelectionSlideId,
                              ga.Id, afmCollected, this.m_Sheet.Ink,
                              this.afmSelection, this.m_Control.FindForm());

      (continued next page)
      (continued from previous page)

               if (g)
               {
                  this.m_Sheet.Selection = null;
               }
               CPresenter_updateListeners(this.m_Sheet);
            }
         }
      }
   }
}
```

The *IStylusSyncPlugin.CustomStylusDataAdded* is triggered when custom data is available. When that custom data consists of a gesture the *if* statement above will be executed. In short, the gesture is identified (as a "check", "circle", etc) and, if the lasso was used to trigger the gesture (*afmcollected.Count != 0*), then all the relevant locks are applied and the method *exec_gesture* from the *MST library* is called. After all the actions triggered by the gesture are completed, all the listeners of the current session receive the updated screen.

With respect to the structured selection, the following code was added:

```
void IStylusSyncPlugin.SystemGesture(RealTimeStylus sender,
                                     SystemGestureData data)
{
   (...)

   // Double tap
   if (data.Id == SystemGesture.DoubleTap)
   {
      taps = 2;

      using (Synchronizer.Lock(this.m_Display.SyncRoot))
      {
         using (Synchronizer.Lock(this.m_Sheet.SyncRoot))
         {

      (continued next page)
```

```
    (continued from previous page)

            Point x = new Point(lastX, lastY);
            Node root = MSTVariables.structSlides.get(
                                                this.m_Display.Slide.Id, x);

            if (root != null)
            {
                int sncount = selectedNodes.Count;
                MST.Structure.Search.FullTreeSelection(
                                selectedNodes, root,
                                x, this.m_Sheet.Ink.Strokes);
                Rectangle r;
                for (int i = 0; i < selectedNodes.Count; i++)
                {
                    r = ((Node)selectedNodes[i]).BoundingBox;
                    selectedStrokes.Add(this.m_Sheet.Ink.HitTest(r,
                                        MSTVariables.percentIntersect));
                }

                this.m_Sheet.Selection = selectedStrokes;
                this.afmSelection = selectedStrokes;
                this.afmSelectionSlideId = this.m_Display.Slide.Id;
                this.m_Display.Invalidate();
            }
        }
    }
  }
}
```

When a double tap is detected, the node that contains the point of the last tap (*x*) is obtained and used as an argument for the *FullTreeSelection* method from the *MST library*. After its execution, all the strokes contained in the *selectedNodes* are added to the *selectedStrokes* which are then set to be the current selection.

As the two examples above show (structured selection and gestures), the use of the *MST library*'s methods in the *CP* is quite straightforward. However, some small changes had to be done to other parts of the *CP*'s code. For example, some new variables where added to the code to keep values that are needed for the use of the *MST library*'s methods and their values are set while the system is handling packets (*HandlePackets* method). Variables like *afmSelection*, *afmCollected*, *lastX*, *lastY* are new and correspond to, respectively, the last selection that was done, the last sequence of points collected through the use of the lasso, and the *x* and *y* coordinates of the last point where the stylus touched the screen.

MathSpadMenu The *MathSpadMenu* corresponds to a new menu in the user interface, from which the user can start the recogniser, create new structure, manipulate structure, turn on/off the animations and access the editors for the operators and gestures. This class was created following the same structure of the other menus that were already available in the *CP*. For each of the menu options a class was created containing the definition of the constructor for the menu item and definition of the behaviour when the menu item is clicked. For example, for the recognition and creation of structure the following class was created (some code has been omitted for simplicity):

```
public class RecognizeMenuItem : MenuItem, InkSheetAdapter.IAdaptee
{
   protected readonly PresenterModel m_Model;
   private readonly EventQueue m_EventQueue;
   private InkSheetModel m_Sheet;

   public RecognizeMenuItem(ControlEventQueue dispatcher,
                                          PresenterModel model)
   {
      this.m_EventQueue = dispatcher;
      this.m_Model = model;
      this.Text = "Recognise␣and␣create␣structure";
      Misc.MenuShortcutHelper.SetShortcut(this, Keys.Control | Keys.M);
   }

   protected override void OnClick(EventArgs e)
   {
      base.OnClick(e);

      using (this.m_Model.Workspace.Lock())
      {
         using (Synchronizer.Lock(
                 this.m_Model.Workspace.CurrentDeckTraversal.Value.SyncRoot))
         {

      (continued next page)
```

```
    (continued from previous page)

        SlideModel slide =
         this.m_Model.Workspace.CurrentDeckTraversal.Value.Current.Slide;
        InkSheetModel sheet = (InkSheetModel)slide.AnnotationSheets[0];

        using (Synchronizer.Lock(sheet.SyncRoot))
        {
          Microsoft.Ink.Strokes selection = sheet.Selection;
          if (selection != null)
          {
            CallRecognizer cr = new CallRecognizer(selection);
            String c = cr.recognize();
            String bb = cr.BoundingBoxes;
            CallParser cp = new CallParser(c, bb);
            MSTVariables.root = cp.createStructure();
            MSTVariables.structSlides2.add(slide.Id,
                                      MSTVariables.root);
          }
          else
          {
            MessageBox.Show("You␣have␣to␣select␣something!");
          }
        }
      }
    }
  }

  (...)
}
```

The *RecognizeMenuItem* constructor was created similarly to the ones already existing in the code. The *OnClick* method obtains all the locks that are needed to avoid interference from other actions and makes use of the *CallRecognizer* and *CallParser* classes that were mentioned and detailed in section 5.2.2 and chapter 4. First, the *CallRecognizer* is used and its result is passed as an argument to the *CallParser*. The result returned by the *CallParser* is then used to be inserted in its correct position in the tree of the slide where the expression is situated. For all the other menu and sub-menu items, similar classes were created making direct use of the methods provided by the *MST library*.

5.3.2 Conclusions

The use of the *MST library* within the *Classroom Presenter 3.1* has proved to be quite straightforward. Most facilities were used almost directly with only a few small adaptations needed. Of course, without documentation, this integration took a significant time to be done but it could possibly be done much quicker if it was done by the *CP*'s creators.

Also due to the lack of documentation, some features are not implemented as in the original *CP* implementation. For instance, it was difficult to update the listeners' (students) slides when ink was changed. In the *CP*, ink is seen by the listeners in real-time, but in this adaptation only the final results of the use of gestures and animations are seen on the listeners' side. Having more knowledge of how to update the listeners should make it possible to overcome this problem. However, animations are still useful since they can still be seen in the public display that everyone in the classroom should be able to see.

In conclusion, the integration of the *MST library* into the *Classroom Presenter* supports the point that the library is usable and easy to use within other projects.

CHAPTER 6

Evaluation and testing

In this chapter, the results of the tests carried out to evaluate the functionalities of the *MST library* are presented. The tests are divided in two parts: teachers' testing and students' testing.

Teachers' testing consists of a combination of usability and suitability testing whilst students' testing concentrates only on suitability evaluation. Details about each part, as well as a discussion about the results, are given in the following sections.

6.1 Software evaluation

Software can be evaluated for many purposes: detection of bugs, usability testing, suitability evaluation, performance testing, etc. Although detection of bugs and performance testing are crucial in industry, they may not be crucial in some other contexts. In the scope of a research project, for instance, usability testing and suitability testing can be more important than performance testing. As in the scope of the *MST library*'s project the goal is not to provide a professional tool but, instead, to provide a prototype that exemplifies new ways of dealing with mathematical content and presenting it to an audience, usability and suitability testing are important.

The aim of the evaluation of the *MST library* is to determine if its functionalities are usable and whether or not they are suitable for the goals of the project. The testing performed is based on use cases representing common interactions between users and the system. The usability testing is focused on teachers as they are the intended main users of the system. It was carried out following the suggestions in [Rub94]. The suitability testing [Has08, Chapter 5] aimed at determining whether the system is suitable for helping the users execute their tasks. Normally, when performing suitability testing, users interact directly with the system. In this case, teachers interacted directly with the system, but students did not, because the idea was to emulate the situation that occurs more often in practice: the teacher delivers some mathematical content and students observe and take notes. Suitability was measured through the use of questionnaires.

As it will be clear in the following sections, the results obtained are positive and seem to indicate that the functionalities provided are "fit for purpose" and usable.

6.2 Evaluation by Teachers

The main purpose of this part is to record the teachers views on the usefulness of the product as a teaching aid, the difficulties found whilst using the software and their general opinion about the software's features. The number of testers was small and, as such, the results do not provide definite answers. However, they provide an indication of the system's usability and usefulness.

Problem Statements:

1. Is the software usable?
2. Do users think that the software is useful for teaching?
3. What difficulties did they find whilst using the system?
4. Which features did they find useful?
5. Is there any relation between the users' opinion about the calculational method and their opinion about the software?
6. What are the users' suggestions for improving the software?
7. What is their opinion about the technology used (pen versus touch and pen versus keyboard)?

User Profile: Teachers and PhD students that have been exposed previously to the calculational method. Some users use the calculational method to teach and some have only been exposed to it in a course. Most users have never used a Tablet PC.

Number of users: 8 (eight).

Methodology: Users were given a guide (see Appendix A) detailing some steps that they had to follow in order to write a proof using the system. A questionnaire (see Appendix B) was given to record their opinion. Also, the users were observed by a monitor.

6.2.1 Results

Is the software usable?
To determine whether the software is usable or not, users were given a task guide to follow. The guide detailed a sequence of steps to be performed in order to produce a proof. For each step, the user was asked to record the level of difficulty in using each feature involved. The results obtained clearly indicate that the level of difficulty decreased as the users got familiar with the software. The level of difficulty reported by most users at the first use of each feature decreased with further uses. The monitor noticed that during the first minutes of the test, most users struggled to perform the tasks[1]. However, they soon got used to the software/pen and performed the remainder of the test without problems.

In table 6.1 the average results reported for the features used during the task are presented[2]. For features that had multiple uses, the results for the first and last use are reported. For the remainder, the results for the first use are shown. The level of difficulty was expressed using a scale from zero (Very easy) to four (Very difficult)[3].

For the *Selection* and *Copy* features it is clear that the difficulty decreased. For *Leibniz*, although only slightly, the level of difficulty increased. However, it is still reported as easy to perform. This increase can possibly be linked to the respective steps of the proof: *Leibniz* is triggered by drawing a check over a symbol; the first point of the check is used to determine which symbol should be substituted; in the last use of *Leibniz* the symbol to be substituted is much smaller than in the first use making it more difficult to target it. All the other features were regarded as *easy* or *very easy* to use at the first attempt.

[1]Some problems were due to the users being unfamiliar with the use of Tablet PCs

[2]The average results are rounded to two decimal places

[3]In the actual test the scale used was the inverse: 0 (Very difficult) to 4 (Very easy). This scale was inverted to facilitate the exposition of the results.

Table 6.1: Level difficulty reported(0 - Very easy; 4 - Very difficult)

Feature	First use	Last use
Selection	2.00	0.75
Copy	1.38	0.50
Leibniz	0.50	0.63
Add brackets	0.38	n/a
Distributivity	1.00	n/a
Remove brackets	0.50	n/a
Ungroup	1.00	n/a

Figure 6.1 shows the average of the level of difficulty reported by the users for each step of the proof. It is clear from the graphic that the level of difficulty decreased as the users got familiar with the software. Moreover, the level of difficulty never exceeded the level of 2 (two) which seems to indicate that the software is usable.

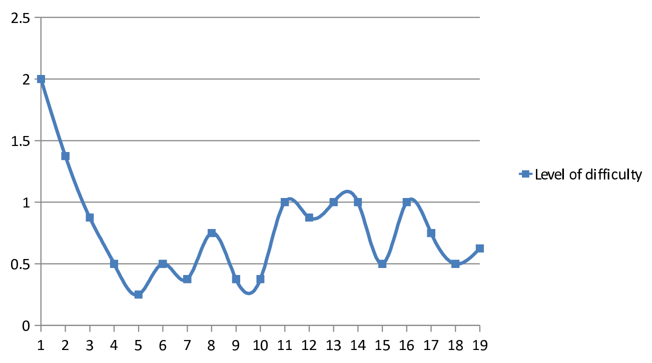


Figure 6.1: Level of difficulty in using the features (0 - Very easy; 4 - Very difficult)

When asked how easy it was to complete the task using the software, the average result was 0.88 meaning that, on average, the users found it easy to use the software.

Do users think that the software is useful for teaching?

Users were asked to comment on the statement "The features provided by this software can help with teaching mathematics". In figure 6.2 the results are presented[4].

Clearly, the users agree with the statement (5 (63%) users do agree and 3 (38%) do

[4]Some of the percentages shown in this chapter do not add up to 100%, due to the rounding. For example, percentages 62.5% and 37.5% will be shown as 63% and 38%.

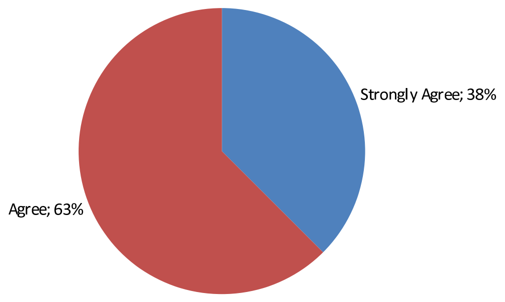


Figure 6.2: Statement: The features provided by this software can help with teaching mathematics

strongly agree). No one disagrees.

What difficulties did they find whilst using the system?

Users were asked about the difficulties they found when using this software. Three users reported that they had no particular difficulties apart from getting used to the software initially. One reported difficulties with performing taps in the first attempt but it became very easy. One reported problems with selecting substructures but indicates that the problem could have been their inexperience with tablets. One user reported problems with taps and difficulty in knowing if "invisible" actions had been applied ("invisible" in the sense that the change in structure does not change the ink, e.g. associativity) — however, it can be easily checked by using taps. Another user reported problems with gesture recognition, accessing the menu, and implicit binding of operators (again, in the sense that it is not visible — for instance, the precedence of operators). One user reported problems with using distributivity.

Overall, users did not seem to find many difficulties in using the system. However, as two users report difficulties with "invisible" structure, it may be useful to improve it in the future.

Users were also asked to indicate what they disliked the most about the software. Answers included the inability to move text (1 user), selection did not always work (2 users), bad gesture recognition (1 user), and the use of menus to apply certain actions (which was indicated by 4 users). The results indicate that the main dislike is regarding the use of menus for some actions. That should be improved in future versions.

Which features did they find useful?

Users were asked their opinion about some specific features. When asked if they thought that animations would be useful for teaching, 6 (75%) users answered *Yes* whilst only 1 (13%) answered *No*; 1 (13%) did not provide an answer. Figure 6.3 shows the results. Clearly, animations are seen as a useful feature.

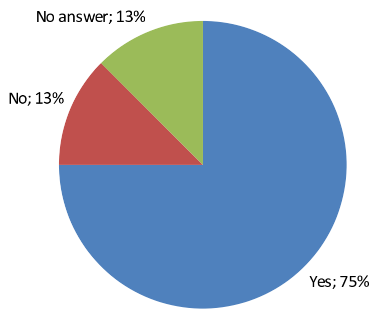


Figure 6.3: Do you think that animations would be useful for teaching?

Users were also asked if a play/pause feature for the animations would be useful (figure 6.4): 6 (75%) users answered *Yes*; 2 (25%) answered *No*.

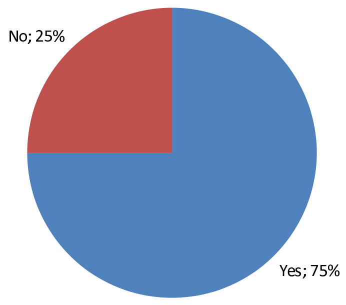


Figure 6.4: Do you think that play/pause features for the animations would be useful?

Regarding the use of gestures to trigger actions, users clearly like it (figure 6.5). When asked their opinion on the statement “Gestures are a good way to trigger actions”, all the opinions were positive (7 (88%) users strongly agree and 1 (13%) agree).

When asked if they prefer to manipulate formulae through structure manipulation rather than by moving ink, users definitely preferred the structure manipulation: 5 (63%) users strongly agree with the statement “I prefer to manipulate a formula through

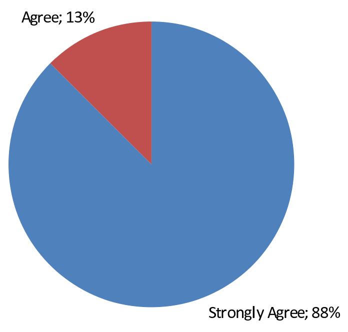


Figure 6.5: Statement: Gestures are a good way to trigger actions

the structure than manually moving the ink around to obtain the desired result" and 3 (38%) agrees with the statement.

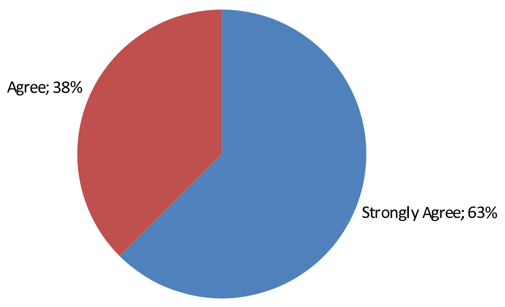


Figure 6.6: Statement: I prefer to manipulate a formula through the structure than manually moving the ink around to obtain the desired result

Users were also asked to indicate what they liked the most about the software. The answers included accuracy of copy, using a gesture to apply distributivity, applying distributivity automatically, showing the dynamics of formulae manipulation, simplicity, and intuitive usage.

Is there any relation between the users' opinion about the calculational method and their opinion about the software?
Given that every user had a positive opinion about the calculational method, it is not possible to draw a conclusion. However, as the overall opinion about the software is positive, this can suggest that users that like the calculational method also like the software.

What are the users' suggestions for improving the software?
The suggestions include a menu displaying a list of possible selections when attempting to select, display of syntax tree, making "invisible" syntax visible by using colours, add more icons to the menu to trigger the actions available currently in a sub-menu, add ability to select and move text, link gestures to a sequence of actions, use auto-complete for some structures, and manipulation of brackets by using gestures.

What is their opinion about the technology used (pen versus touch and pen versus keyboard)?

Figure 6.7 shows the teachers opinions on the statement "I prefer to use the handwritten representation of formulae than their representation in typeset". 3 (38%) users have no preference between the handwritten representation of formulae and its typeset representation; 2 (25%) users have a preference for the handwritten representation; 1 (13%) user prefers the typeset representation; 1 (13%) strongly prefers typeset; 1 (13%) user did not provide an answer. These results seem to suggest that there is no strong tendency to a type of representation.

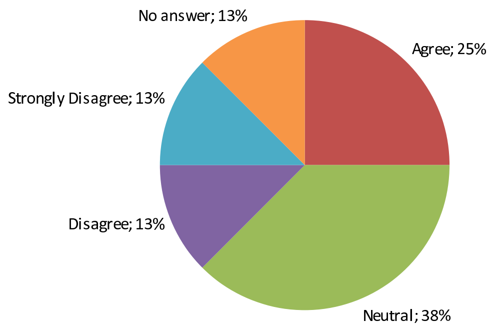


Figure 6.7: Statement: I prefer to use the handwritten representation of formulae than their representation in typeset

In figure 6.8 the teachers opinions about the statement "I find it easier to handwrite mathematics than writing it using a keyboard" are presented. 3 (38%) have a strong preference for handwriting mathematics; 2 (25%) users have a preference for handwriting mathematics; 1 (13%) user has no preference; 1 (13%) strongly prefers the use of a keyboard; 1 (13%) did not provide an answer. These results suggest that handwriting mathematics is preferred to the use of a keyboard.

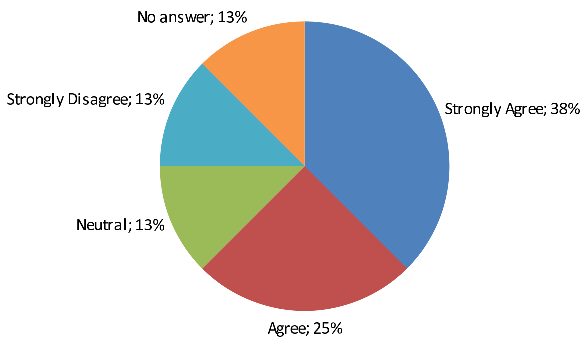


Figure 6.8: Statement: I find it easier to handwrite mathematics than writing it using a keyboard

The opinions of the users about the statement "Input using a pen is more precise than using the fingers" are presented in figure 6.9. 2 (25%) users strongly agree with the statement; 3 (38%) do agree with it; 1 (13%) disagrees; 1 (13%) strongly disagrees; 1 (13%) user is neutral. From these results it can be concluded that pen input is seen as more precise than the use of touch.

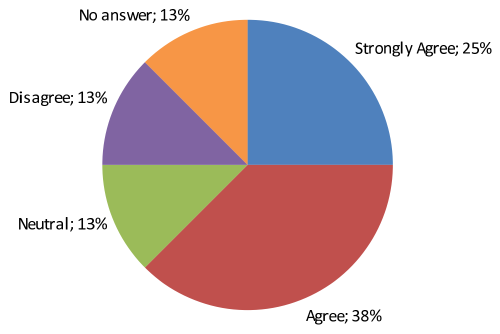


Figure 6.9: Statement: Input using a pen is more precise than using the fingers

6.3 Evaluation by Students

The main purpose of this part is to determine whether students find the features proposed useful for learning, to check which features are most useful, and gather suggestions for improvement. The test is also designed to determine a relation between the students' liking of the calculational method and their opinion about the product.

Problem Statements:

1. Do students think that the software is useful for learning?
2. Which features did the students find more useful? What is the opinion of the students about some of the features (in particular, gestures, interaction with the teacher, and animations)?
3. Is there any relation between the students' opinion about the calculational method and their opinion about the software?
4. What are the students' suggestions for improving the software?
5. What is their opinion about the technology used (pen versus touch and pen versus keyboard)?

Students profile: Students that have been exposed previously to the calculational method in a second-year undergraduate module on program calculation.

Number of students: 74 (seventy four).

Methodology: The test consisted of a 20 minutes presentation followed by an anonymous questionnaire (see Appendix C). The presentation started with an introduction to the system and an overview of its features. Also, some details about the questionnaire were given and some terms were explained to avoid confusion. In the end, the system was used to write a calculational proof as it would be done in a lecture. The proof demonstrated was the one presented in Appendix A.

Students did not use the software, since the main goal of the tool is to be used as a teaching and presentation aid by teachers. The idea was to emulate the situation that occurs more often in practice: the teacher delivers some mathematical content and students observe and take notes.

6.3.1 Results

Do students think that the software is useful for learning?
When asked if the software helped understanding the manipulations involved in the

proof that was demonstrated, 63 (85%) students answered *Yes* whilst only 8 (11%) answered *No*; 3 (4%) students did not answer. This shows that the software has definitely helped the students understanding the steps of the proof. Figure 6.10 illustrates the results.

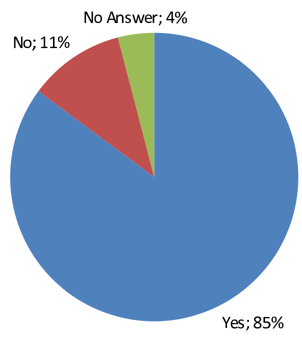


Figure 6.10: Did the software help you understanding the manipulations involved?

The opinions of the students about the statement "This software can help me doing calculations by myself" are presented in figure 6.11: 4 (5%) students strongly agree with the statement; 46 (62%) agree; 19 (26%) are neutral whilst 5 (7%) disagree with the statement. Clearly, most students think that the software can help them with writing calculations. However, we cannot conclude that the software helps them with writing proofs, since the students did not use the software.

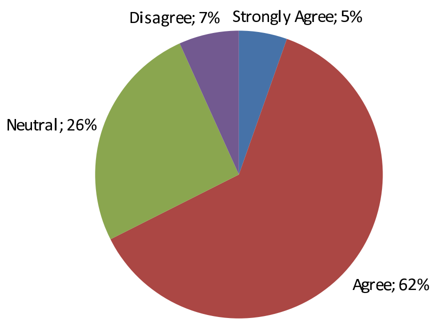


Figure 6.11: Statement: This software can help me doing calculations by myself

In figure 6.12 the students' opinions about the statement "This software can improve my experience in the classroom" are presented. Most students (49 (66%)) agree with the statement; 11 (15%) strongly agree; 10 (14%) are neutral and only 3 (4%) disagree; 1

(1%) student provided an invalid answer.

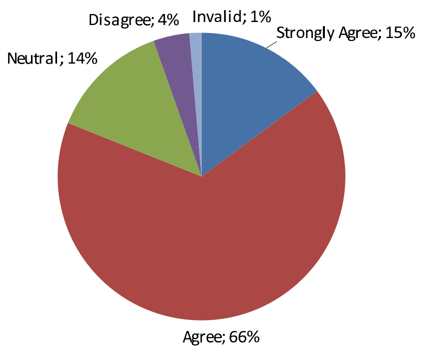


Figure 6.12: Statement: This software can improve my experience in the classroom

Regarding their opinion about the statement "This software can help me with learning calculational mathematics" the opinions are still positive: 6 (8%) students strongly agree with the statement; 46 (62%) agree; 10 (14%) are neutral and the same number (10 (14%)) disagree; 1 (1%) strongly disagrees and 1 (1%) provided an invalid answer. Figure 6.13 illustrates these results. Although the majority of students say that the software can help, there is still room for improvement.

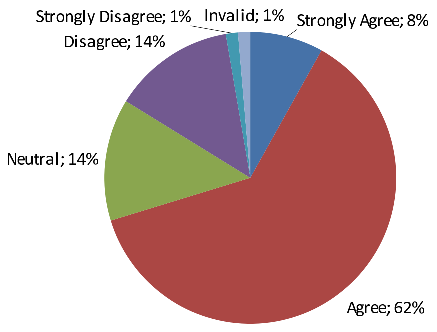


Figure 6.13: Statement: This software can help me with learning calculational mathematics.

To determine if the software is seen as useful for learning mathematics in general, students were asked their views on the statement "This software can help me with learning general mathematics". The results were not as positive but were still good. The majority of students agrees with the statement: 30 (41%) agree and 7 (9%) strongly agree; however, 27 (36%) are neutral which is a great increase compared with the previous statement; 8 (11%) disagree and 2 (3%) strongly disagree.

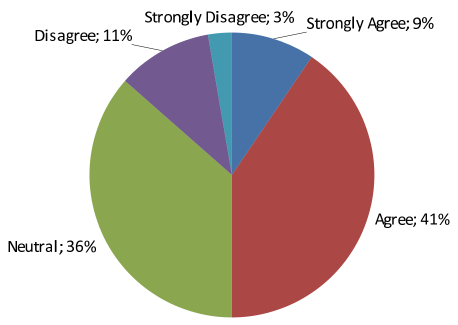


Figure 6.14: Statement: This software can help me with learning general mathematics

Students were also asked if they would have used the software if it became available during their attendance at the Program Calculus course (a course involving calculational mathematics). The results are illustrated in figure 6.15. The great majority of students answered *Yes* (65 (88%)); only 7 (9%) answered *No*; 1 (1%) student did not answer and 1 (1%) provided an invalid answer. Definitely, students found the software useful for calculational mathematics.

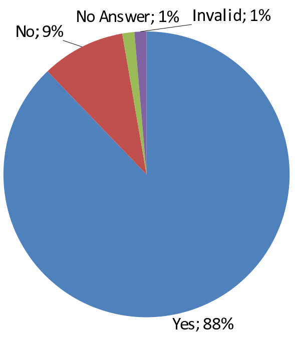


Figure 6.15: If this software became available during your attendance to the Program Calculus course, would you have used it?

When asked if they would use the software for other purposes, 38 (51%) students answered *Yes*, 28 (38%) answered *No*, and 8 (11%) did not provide an answer (figure 6.16). Students provided some suggestions for other uses: verification of proofs including proofs from digitalised handwritten documents, for use with interactive whiteboards, for mathematics in general, and physics.

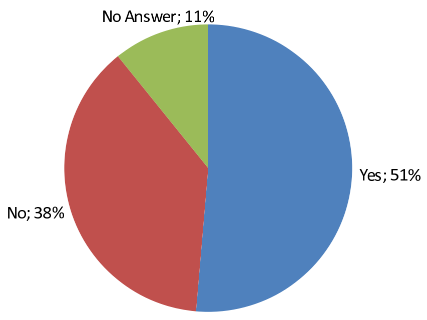


Figure 6.16: Would you use this software for anything else?

Which features did the students find more useful? What is the opinion of the students about some of the features (gestures, interaction with the teacher, and animations)?

Although the ability to communicate with the teacher's computer is not a feature provided by the software developed by the author, students were asked their opinion about it. The goal was to assess if the belief of the author that the feature is useful was true. Students were asked to comment on the statement "Being able to send answers from my computer to the teacher's makes it easier to participate in the lectures". Figure 6.17 shows the results: 31 (42%) students strongly agree with the statement; 28 (38%) agree; 12 (16%) students are neutral and 3 disagree (4%).

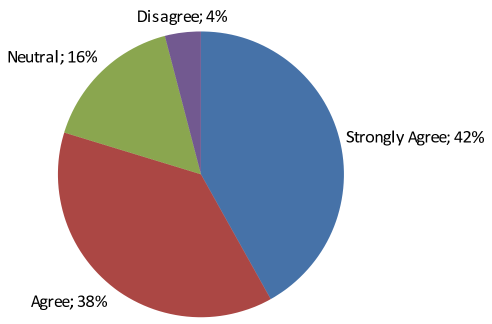


Figure 6.17: Statement: Being able to send answers from my computer to the teacher's makes it easier to participate in the lectures

They were also asked to give their opinion about the statement "Being able to send answers from my computer to the teacher's makes the lectures more interactive" (figure 6.18): 40 (54%) students strongly agree, 24 (32%) agree, 8 (11%) are neutral, and 2 (3%)

disagree.

Thus, it is safe to conclude that the ability for the students to send their answers directly to the teacher's computer during the lecture is considered useful.

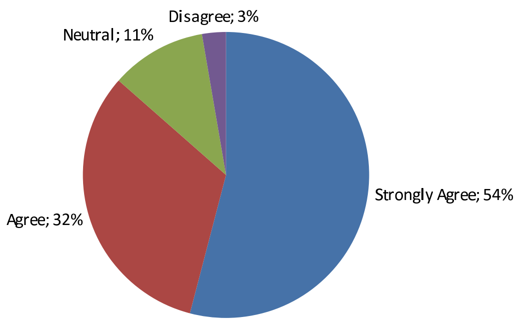


Figure 6.18: Statement: Being able to send answers from my computer to the teacher's makes the lectures more interactive

Questions about the animations feature were also included — during the presentation, the animation for the distributivity rule was shown. Students were first asked if they think that animations would be useful to understand better how algebraic rules are applied. Clearly students see animations as useful since 68 (92%) answered *Yes* while only 6 (8%) answered *No*. Figure 6.19 shows these results.

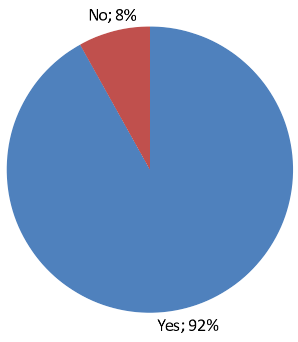


Figure 6.19: Do you think animations would be useful to understand better how algebraic rules are applied?

Regarding the addition of play/pause features to the animations (figure 6.20), 69 (93%) students thought it would be useful and only 5 (7%) thought contrary.

Students were also asked their opinion about the statement "Gestures are a good way

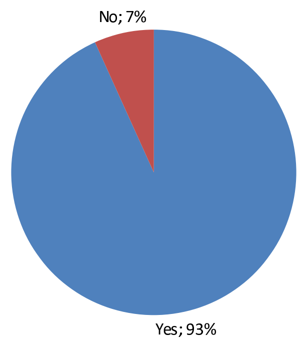


Figure 6.20: Do you think that play/pause features for the animations would be useful?

to trigger actions". Figure 6.21 presents the results: 27 (36%) strongly agree with the statement; 43 (58%) agree; 3 (4%) are neutral and 1 (1%) disagrees. Thus, it can be concluded that students find gestures good as a trigger for actions.

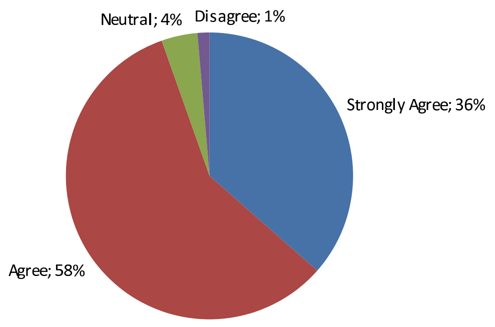


Figure 6.21: Statement: Gestures are a good way to trigger actions

Finally, they were asked to provide their views on the statement "The gestures used during the presentation are adequate to the operations to which they are associated": 19 (26%) strongly agree; 38 (51%) agree with the statement; 14 (19%) are neutral and 3 (4%) disagree. These results suggest that gestures used to trigger the operations used in the presentation (selection, copy, Leibniz, and distributivity) are adequate. Figure 6.22 illustrates the results.

Regarding what they liked the most about the software, the most popular answers were simplicity, use of gestures to apply rules, ease to learn, recognition of handwriting, application of rules, interactivity, animations, the gesture for distributivity and handwritten input. Students also indicated the avoidance of repetitive writing, reduction

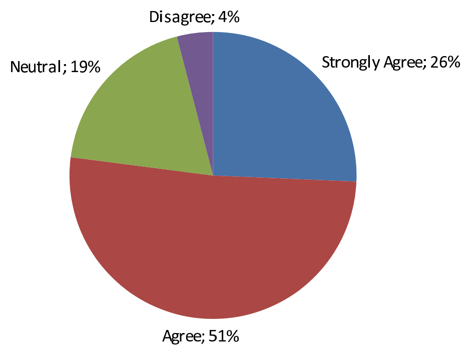


Figure 6.22: Statement: The gestures used during the presentation are adequate to the operations to which they are associated

of human errors in proofs, manipulation of expressions, the automatisation of steps in proofs, and the possibility of having several computers following the lecture in real-time.

Students were also asked to point out what they disliked the most about the software. The most popular answers were the graphical interface, the use of menus, the software being oriented to pen-based devices (since most students don't own one), and slowness of animations.

Is there any relation between the students' opinion about the calculational method and their opinion about the software?

To answer this question, the average of the answers of each student for questions 5 (five) to 9 (nine) was calculated[5] and the result was combined with their opinion about the calculational method. This result indicates that 27 (36%) students have a positive opinion about the calculational method and the software; 1 (1%) student has a positive opinion about the calculational method but was negative about the software; 2 (3%) have a positive opinion about the method but are neutral regarding the software; 4 (5%) have a negative opinion about the method but are positive about the software; 1 (1%) has a negative opinion about both the method and the software; 36 (49%) have a neutral opinion about the method and a positive opinion about the software; finally, 3 (4%) have a neutral opinion about the method and a negative opinion about the software. Figure 6.23 illustrates these results. From these results it is not possible to draw a conclusion — given that the majority of users that liked the software have a neutral

[5]The average gives an indication of the students' opinion about the software.

opinion about the method.

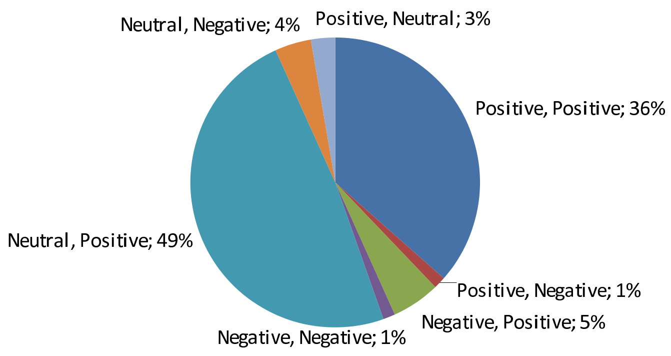


Figure 6.23: Relation between the students' opinion about the calculational method and the software (method,software)

What are the students' suggestions for improving the software?

The students' suggestions include improving animations, connection with theorem provers, replay feature for sessions, gesture to add/remove brackets, clickable list of rules, suggestion of applicable rules, possibility to hide and expand steps in proofs, improve appearance of the interface, and further develop the system so that students can receive support from the teacher when away (including chat/video calls).

What is their opinion about the technology used (pen versus touch and pen versus keyboard)?

In figure 6.24 the students' opinions about the statement "I find it easier to handwrite mathematics than writing it using a keyboard" are presented: 49 (66%) have a strong preference for handwriting mathematics; 16 (22%) students have a preference for handwriting mathematics; 7 (9%) students have no preference; 2 (3%) have a preference for using a keyboard and 1 (13%) strongly prefers the use of a keyboard. These results clearly show that handwriting mathematics is preferred to the use of a keyboard.

The opinions of the students about the statement "Input using a pen is more precise than using the fingers" are presented in figure 6.24. 37 (50%) students strongly agree with the statement; 25 (34%) do agree with it; 11 (15%) students are neutral; 1 (1%) disagrees. From these results it can be concluded that pen input is regarded as more precise than the use of touch.

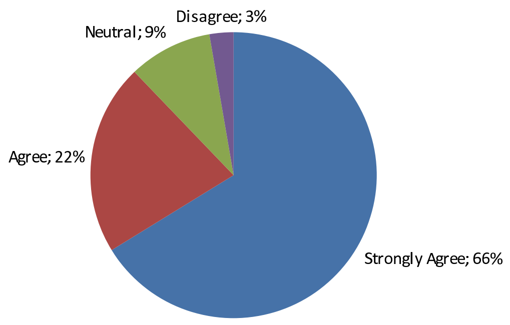


Figure 6.24: Statement: I find it easier to handwrite mathematics than writing it using a keyboard

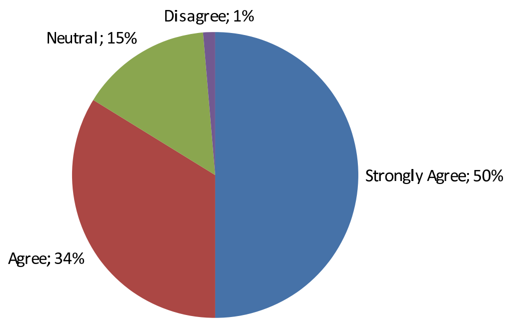


Figure 6.25: Input using a pen is more precise than using the fingers

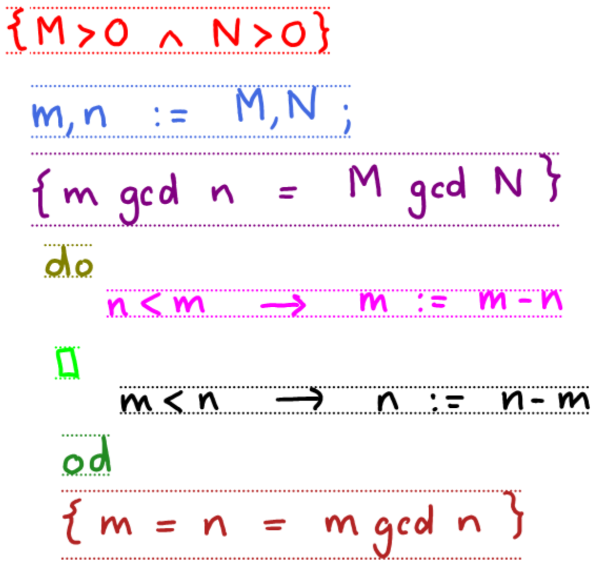


Figure 6.26: Sample 1: Line division (algorithm 1)

6.4 Testing the line division algorithm

The last task that the teachers had to perform using the tool was to provide two handwritten samples (see task 3, Appendix A). The samples were used to test the line division algorithm (algorithm 1, page 42) against the *Microsoft*'s *Ink.Divider*. The algorithm 1 clearly outperformed the *Ink.Divider*.

For the first mathematical text, the *Ink.Divider* returned the wrong division for every single sample; algorithm 1 returned 5 (five) correct and 3 (three) incorrect results. The incorrect results are due to a mismatch between the users' handwriting and the expected input format. The algorithm assumes that strokes from one line do not intersect the bounding box of another line. If the users knew this beforehand, the results would probably be better. However, users were not informed of this assumption since it would influence their samples and the intention was to test both algorithms with the users' usual handwriting.

The following images show some of the samples given by the users. Each colour represents a line as detected by the division algorithms. For algorithm 1, dotted lines are also used to show the bounding box of each line.

In figure 6.26, the result obtained by the algorithm 1 for that particular sample is shown. The user provided a very neat input resulting in a correct division. Clearly, there are no intersections between lines and strokes that are part of the same line are clearly aligned.

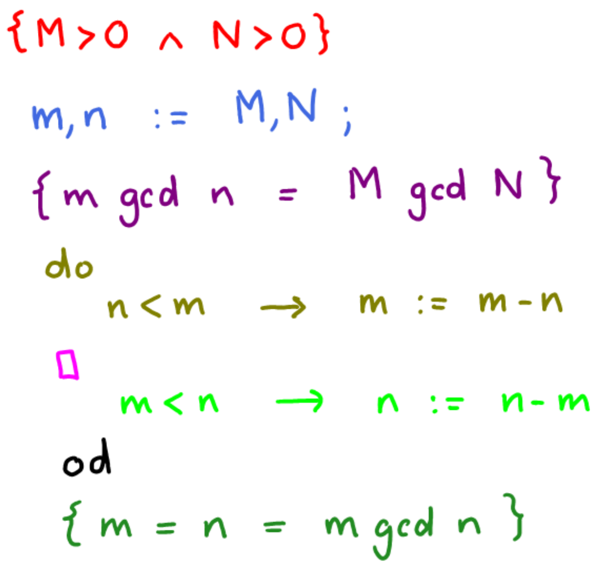


Figure 6.27: Sample 1: Line division (Ink.Divider)

However, the *Ink.Divider* did not provide the correct division because the command “do” was considered part of the line below (see figure 6.27). Even though the handwriting is neat, the *Ink.Divider* was uncapable of determining the right result.

Figure 6.28 shows the line division returned by algorithm 1 for a second sample. In this case, the division is incorrect. The reason is the clear intersection between lines as shown in figure 6.29 (red lines show where the intersections occur). The algorithm determines iteratively which strokes intersect the current bounding box of the line; every time a new stroke is detected, it is added to the current line and the new bounding box (including the newly added strokes) is used for another iteration. For this sample, the commas of the second line force the inclusion of the last curly bracket of the following line (given that they intersect vertically, i.e, they have ’y’ coordinates in common); as the bounding box is extended to include this curly bracket, the following iteration captures all the strokes that are part of the third line; at this point, the leftmost ’g’ is included in the current line and forces the capture of the ’d’ in the fourth line, which, in turn, captures the ’o’. In the end, what should have been returned as second, third and fourth lines, is now part of one single line. Lines six and seven are also considered part of the same line because the symbol ’□’ intersects vertically the last ’m’ of the following line.

Regarding the second mathematical text, the *Ink.Divider* returned 1 (one) correct result and 7 (seven) wrong; algorithm 1 got 7 (seven) correct results and only 1 (one) wrong

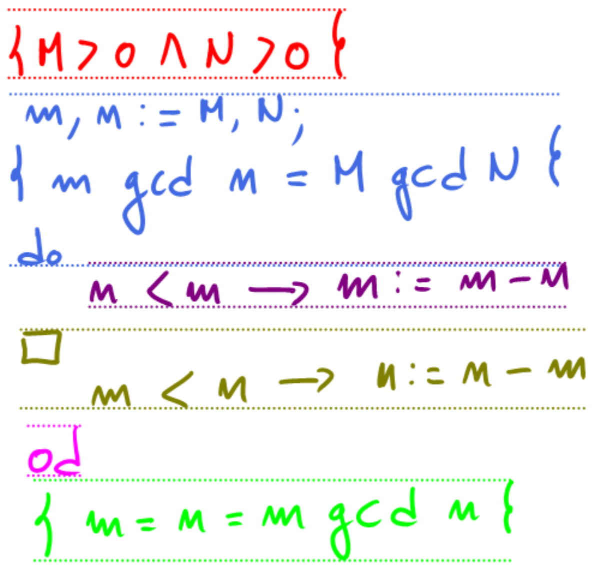


Figure 6.28: Sample 2: Line division (algorithm 1)

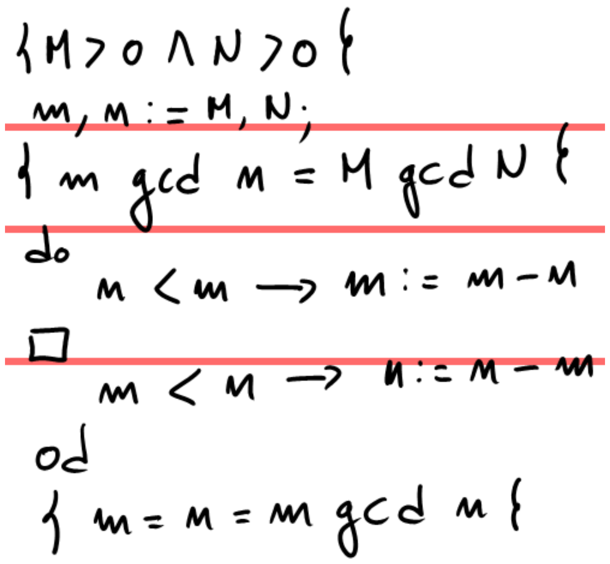


Figure 6.29: Sample 2: Line division (algorithm 1) – intersecting lines

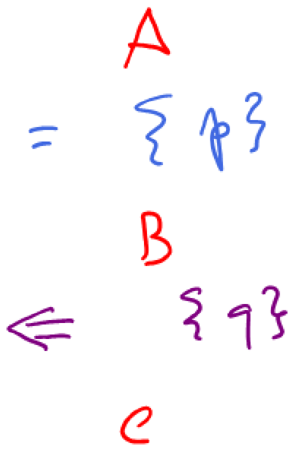


Figure 6.30: Sample 3: Line division (Ink.Divider)

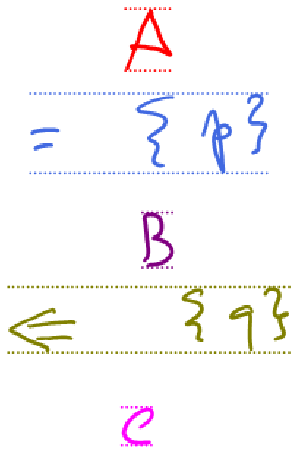


Figure 6.31: Sample 3: Line division (algorithm 1)

division.

In figure 6.30, a division done by the *Ink.Divider* is shown. The result is incorrect because 'A', 'B', and 'C' are seen as part of the same line. It looks as though the algorithm found a vertical pattern and assumed there was a vertical line.

On the other hand, algorithm 1 returned the correct division (figure 6.31). Clearly there are no intersections between lines and the algorithm behaves as expected.

However, algorithm 1 failed for the sample presented in figure 6.32. Again, the problem was the vertical intersection between lines: the left curly bracket in red intersects 'A' and the right curly bracket in blue intersects 'B' and 'C'.

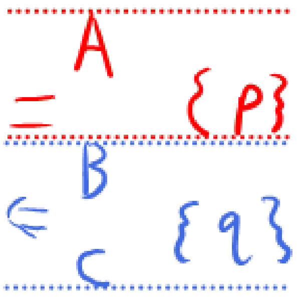


Figure 6.32: Sample 4: Line division (algorithm 1)

6.5 Conclusions

The tests were designed so that teachers would experiment with the software as a teaching tool and the students would experience it as a learning tool. Students did not use the software, since the main goal of the tool is to be used as a teaching and presentation aid by teachers. The idea was to emulate the situation that occurs more often in practice: the teacher delivers some mathematical content and students observe and take notes.

The results obtained seem to indicate that the software is both usable and suitable for its purpose. They also indicate that the features are not immediately easy to use but become easy with little practice. Also, teachers find the software useful for teaching and students find it useful for learning. However, there are still improvements to be made, as 11 (eleven) students out of 74 (seventy four) — 15% of students — found the system unhelpful for learning calculational mathematics. Possibly, a better user interface could improve these results. The use of gestures to trigger actions was well received by both teachers and students as well as the animations. It also became clear that the features are seen as useful for purposes other than teaching calculational mathematics. The use of handwritten input is preferred by most teachers/students and pen input is regarded by most as more precise than input by touch.

Regarding the line division algorithm, the results show that, for the samples provided, the algorithm 1 outperforms the *Ink.Divider*. Although the results are very poor using the *Ink.Divider*, its accuracy can possibly be improved by adapting its default RecognizerContext to calculational mathematics[6]. However, since the goal of this project was not centered on recognition, the author preferred to use algorithm 1 for its simplicity

[6]The RecognizerContext contains a factoid property that provides context information for ink within a particular field.

and reasonable level of accuracy.

CHAPTER 7

Discussion and conclusions

The objective of this project was to design a structure editor of handwritten mathematics oriented to Tablet PCs. The motivation was that such a system would help with teaching the mathematics used in algorithmic problem solving and, in general, with presentations containing mathematical content. This would be achieved by providing facilities to manipulate the structure of mathematical expressions and by providing features that could facilitate the process of doing mathematics. In particular, one of the goals was to give special attention to features that could show the dynamics of the symbols involved in mathematical calculations. Towards these goals, a library that provides these features was created. The library was used to extend an existing editor and, based on the results presented in chapter 6, the result seems to be an improvement on what was available before and can improve the experience of both the teacher and the students in the classroom.

The novelty of the results of this project is centered on the structured representation of handwritten mathematical expressions which, in turn, makes it possible to implement other novel features like structured selection and the manipulation of handwritten mathematical expressions. The emphasis on using gestures to apply algebraic rules is also new (although other systems support gestures to, for example, evaluate mathematical expressions, these systems are not concerned with the process of writing proofs or algorithms). Other results are also important and add genericity to the system: the possibility to add and edit operators in runtime is useful and the change of associations between gestures and actions (also in runtime) is a novelty as well (to the best of the author's knowledge). Furthermore, the creation of handwritten templates is also new and valuable in an environment where recognition fails often. Handwritten templates are also useful to store complicated mathematical structures, so that the users do not have to go through the recognition and parsing of the structure every time they need to use those.

However, the library has still some problems that need to be addressed. Although not an integral part of the library, the recognition of the handwritten mathematical expressions was a major problem throughout this project and remains an issue that needs to be solved to make the library usable in real teaching scenarios. Without a good recogniser, the recognition of mathematical expressions takes a long time to complete and is very inaccurate, which is a serious limitation in terms of usability. In fact, the recognition is a major handicap of the usability of the *MST library* in real-life situations. However, after the steps of recognition and parsing are done successfully, the features of the library can be used without problems. If the users are not interested in recognising mathematics during the presentations, and prepare the material before the lectures, the system becomes quite usable.

The lack of a suitable recogniser was also a problem throughout the development of the library, since it made it more time-consuming to test the features and develop new ones. The library had to be reorganised to allow the continuation of the project without a suitable recogniser. Although it was never a goal of the project, a recogniser could have been created. However, due to time restrictions, it was not possible to create one. Nevertheless, an attempt was made to improve the recognition module of the system. In May 2009, an 8-month project was started in Portugal with the aim of adapting one of the existing recognisers (more precisely, the recogniser within the *StarPad* system) to the needs of the library. However, no improved results were obtained and the problem remained the same. This attempt was made in the context of the *MathIS* project [mat09], which aims to exploit the dynamics of algorithmic problem solving and calculational reasoning in both maths education and the practice of software engineering. The *MST library* was also created in the context of the *MathIS* project and it has contributed to its goals by providing features that can help with teaching in mathematics education. More information on the MathIS project is presented in [FM09] and [FMBB09].

To conclude the discussion on the recognition problems, it has to be mentioned that there may exist proprietary recognisers that perform better than the recogniser used during the development of this project. However, the author found little interest from owners of proprietary mathematics recognisers in allowing access to their recognition engines. Maplesoft was the only company that gave access to a recogniser; their recogniser was used throughout the development process. However, without support and documentation for the system, it was not possible to make it perform well and according to the needs of the *MST library*.

Although the author has tested the system with many examples throughout the development, and some tests were done with teachers and students, the library can still benefit from more thorough studies. Nevertheless, the feedback received for this work as it currently stands was positive and the suggestions gathered can be used for future improvement.

During the development of the library, the features were tested by the author of this work through a demo system that served to try concepts like the structure and the use of gestures. However, this demo was far from being usable in real teaching situations. In 2009, the author of this work learnt about the existence of the *Classroom Presenter* and, as it contains features that point in the same direction of the *MST library* (i.e, it also explores the use of Tablet PCs in the classroom), from this point in time, the idea of providing a class library made even more sense. At this stage, the focus of the project became solely the library (the demo was only used as a test-bed for concepts and was abandoned later). With the integration of the *MST library* into the *Classroom Presenter*, it became clear that the library is usable within other tools and that its features can have a positive impact on the teachers' and students' experiences. Moreover, the result of this integration shows that most of the initial goals of this project were achieved: the extended *Classroom Presenter* consists, effectively, of a structure editor of handwritten mathematics. Also, it was at this point that it became possible to do some testing with users and targeted audience.

Finally, another problem is the likely existence of programming errors. Further testing can help with detecting and fixing some of them. This project was done and tested for programming errors by only one person and it was done within a small theory research group. The lack of expertise in the area within the group made it more complicated to achieve a more mature tool. It should be noted, however, that it was never the goal of this project to provide a professional tool. The goal was to create a prototype that can be used as a base to develop a professional system.

Although there are limitations in the current system, it can still help improve the experience of teaching and learning mathematics, particularly calculational mathematics. Based on the tests performed, the features provided seem to be useful for both teachers and students. Most of the ideas implemented by the library came from real teaching experience from the supervisor and the author of this project, and also from teaching studies in which the author has participated (for example, [FM09] and [FMC+11]).

7.1 Current system

As mentioned in section 1.3, one of the goals of this project is to provide support for some of the teaching scenarios for algorithmic problem solving presented in [Fer10]. In chapter 1, it was shown how the features developed could help with the *Scenario 1* shown in [Fer10]. Particularly important in that example was the accurate copy of expressions and the manipulation rules for distributivity and symmetry. Moreover, in chapter 5, it was shown how the features could assist with the manipulation of an algorithm. The algorithm used was Euclid's algorithm, which is the main object of study in *Scenario 11* of [Fer10]. In that scenario, there is a proof of a theorem that is particularly useful to demonstrate the manipulation rules provided by the *MST library*, because it depends on the accurate copy of expressions, on Leibniz, on addition and deletion of brackets, on grouping of expressions, and on the distributivity rule. That proof is used in Appendix D to demonstrate how the current system can be used to do mathematics.

7.2 Future work

In this section some suggestions of work that could be done to improve the *MST library* are given. The section ends with ideas to improve the extended version of the *Classroom Presenter* described in chapter 5.

7.2.1 Improve reliability and usability

The reliability and usability of the current system still have room for improvement. Although the main usability problem of the library is related with recognition of handwritten mathematics, and even though the library was created to be adaptable to users preferences and needs, there are still some features that could possibly be improved. Also, some of the features provided could be made more reliable. The following paragraphs provide more details.

Improve the search for nodes: As it was mentioned in section 4.7.1, the search method for finding a node given a point, has a feature that, in case the point is in between two nodes, the nearest node is returned. This feature is useful since characters like the dot can be difficult to target with a pen, resulting in taps around its bounding box. However, this feature is implemented only for expressions written in one line. Supporting it for expressions written in more than one line is more challenging, because from a node there is direct access to its two neighbours (left and right), but not necessarily to the one that is spatially situated above it. This makes it more difficult to find the nearest node to a point that is situated between two lines. Nevertheless, it would be desirable to have this feature implemented.

Correct division of characters with strokes that do not intersect: The current system is unable to detect which strokes form the same unit when the strokes that compose the unit do not intersect. For instance, when writing the assignment symbol (:=), as the strokes that form this symbol do not intersect, they are seen as separate units which is a problem when parsing the expressions. So that the system works well, the user has to be sure that the strokes intersect, which can be very inconvenient. For this reason, the division of strokes should be improved to detect that strokes that are close together make part of the same symbol. This would improve the usability and reliability of the system.

Improve the user templates: The current system provides an experimental and primitive mechanism for user-defined handwritten templates. This feature was created at a late stage of the project and resulted from the difficulty in recognising more complicated structures (due to the poor recognition support). It turned out to be a very useful feature and a more mature version of the current implementation could improve substantially the usability of the system (at least, until a good recogniser becomes available). A direction to improve this feature is to extend it with some of the Math∫pad ideas. For example, in certain situations, it would be good to change a template and have the changes reflected in the presentation. Also, as mentioned in section 4.7.2, a further direction is to unify the mechanism for user-defined characters with the one for handwritten templates, since the idea behind both mechanisms is similar.

Ink adjustment: In a system where ink can be written freely, it is difficult to determine how the content on the screen is divided. Consequently, it may be difficult to adjust the content when new input is added. For instance, when *Leibniz* is used, the size of the expression may change. Because of the change in size, overlapping of content may occur or too much empty space may be left. For this reason, the current system moves the ink in the line where the manipulation occurred, forwards or backwards, to adapt the previous content to the new one. Currently, this change consists in moving all the ink on the right and in the same line. However, since lines are being defined as y-coordinate ranges, the change can inadvertently affect structures that are written in more than one line. Therefore, the algorithms that do the ink adjustment could be improved.

More usable proof-format: Although the proof format can be saved as a template and used at anytime to write proofs, its current form is not very usable. If one saves the proof format as a template and wants to use as a hint an expression that is already on the screen, they just have to select it and use *Leibniz* to substitute the hint's space by the expression. However, if new content is to be inserted in the hint, the easier choice the user has is to write the content somewhere else on the screen, recognise and create its structure, use *Leibniz* to put it in the hint's space, and then delete the original version of the content. It would be much easier if the system could detect that the user was going to write an hint in the hint's space, and use that input to update internally the hint's content. At the same time, it would be useful if the closing bracket of the hint would move according to the space that the user needs (as it happens in the quantifier example presented in section 4.11.2). A version with these improvements would certainly be more usable than the current one.

Animations: As shown by the tests results, animations are seen as useful and could be improved by adding pause, step-forward, step-back or slow-down features. Also, the current list of moves for some of the manipulation rules could improve (for instance, in a swap, some parts of the animation overlap, as seen in figure 4.18).

Gestures: It would be interesting if the gestures editor provided with the *MST library* allowed the addition of custom gestures. This could perhaps be achieved by using Siger, the Simple Gesture Recognition library. For more details about this library, see [sig05].

7.2.2 Improve the recognition of handwritten mathematics

Probabilistic graphical models: As the library would be more usable if a suitable recogniser was available, one of the next developments could be a recogniser that works well for the notation used in algorithmic problem solving. A possible direction would be to extend an existing symbol recogniser with a probabilistic graphical model that embeds the information needed for the recognition of the mathematics used in algorithmic problem solving.

Training the recogniser: A recogniser capable of being trained to a particular handwriting would improve the recognition process. Moreover, the system could save probabilistic information according to the kind of mathematical patterns written by the user, and use these probabilities to improve the recognition. These probabilities could be used, for instance, to adapt the probabilities used in a probabilistic graphical model according to the current user.

7.2.3 Incorporate feedback and further testing

Although some tests were performed, the library could benefit from more testing and educational studies. Moreover, the ideas and improvements suggested during testing can now be used to improve the library.

The most relevant suggestions during testing were:

- when selecting expressions, show all the sub-expressions that can be selected;
- add ability to show the syntax tree of expressions;
- make "invisible" grouping visible by using colours;
- add more icons to the menu for actions that are currently only available in a sub-menu;
- add ability to select and move text;
- link gestures to a sequence of actions;
- use auto-complete for some structures;
- connection with theorem provers;

- replay feature for sessions;
- show list of applicable rules;
- hide and expand steps in proofs;
- improve appearance of the user interface;
- further develop the system so that support can be given online (including chat and video calls).

7.2.4 Improve the extended Classroom Presenter

The extension done to *Classroom Presenter* described in chapter 5 resulted in a prototype that can animate the features of the *MST library*. The following ideas could improve the prototype and are worth considering:

- improve how manipulations and animations are shown on the students' side;
- remove flickering from animations;
- extend the undo/redo functions to work with the manipulation rules;
- allow changing the colour of expressions (currently, strokes can be written in different colours but, after being drawn, their colour cannot be changed); this feature would be useful for teaching;

7.2.5 Summary

The current version of the library could be further improved in many aspects. However, as it currently stands, it already provides novel features that are regarded as useful for teaching and learning. As such, it does what it was intended to: it provides new concepts for dealing with mathematical content and presenting it to an audience. It is hoped that this work will serve as a basis for a professional system that delivers all the facilities for both teaching and learning algorithmic problem solving.

Definitions

Below, definitions of important terms that are assumed to be known (and are not defined within the body of the thesis) are presented.

Ink In this thesis, the term *ink* refers to *digital ink*. Digital ink is a digital representation of handwriting in its natural form. In a typical digital ink system, a digitizer is laid under or over an LCD screen to create an electromagnetic field that can capture the movement of a special-purpose pen, or stylus, and record the movement on the LCD screen. The effect is like writing on paper with liquid ink. The recorded handwriting can then be saved as handwriting or converted to typewritten text using handwriting recognition technology.

Stroke a trace of ink in handwriting. For example, the strokes of a letter are the lines that make it up.

Bounding box in this thesis, bounding box refers to the *smallest* rectangle that contains a given set of objects. For example, given a set of *strokes* S, the bounding box of S is the smallest rectangle that contains all the strokes of S. Other names for the same concept are *minimum bounding box*, *minimum bounding rectangle* and *enclosing box*.

Slide in thesis, slide is used to refer to a page within a presentation. Sometimes the term *structured slide* is used to denote a slide that contains or can contain structured mathematical expressions.

Form (sometimes used as *Windows Form*) Form is the name used for window in the graphical API included as part of *Microsoft .NET Framework*. Forms are the base units of graphical applications built in the .NET framework.

Point the term point is used to refer to an entity that has a location in a two dimensional space (that is, has an x and a y-coordinate), but has no extent (its height and width are zero).

References

[AAHW04] Richard Anderson, Ruth Anderson, Crystal Hoyer, and Steve Wolfman. A study of digital ink in lecture presentation. In *CHI 2004*, pages 567–574, 2004.

[AAS$^+$] Richard Anderson, Ruth Anderson, Beth Simon, Steve Wolfman, Tammy VanDeGrift, and Ken Yasuhara. Experiences with a Tablet PC based lecture presentation system in computer science courses. In *SIGCSE 2004*.

[ant12] Antlr parser generator (website). URL: `http://www.antlr.org`, 2012. [Last accessed: 15 April 2012].

[Bac01] Roland Backhouse. Mathematics and programming. A revolution in the art of effective reasoning. Inaugural lecture, School of Computer Science and IT, University of Nottingham, October 2001.

[Bac03] Roland Backhouse. *Program Construction*. John Wiley and Sons, Inc., 2003.

[Bac11] Roland Backhouse. *Algorithmic Problem Solving*. John Wiley and Sons, Inc., 2011.

[BH93] R. C. Backhouse and P. F. Hoogendijk. Elements of a relational theory of datatypes. In B. Möller, H. Partsch, and S. Schuman, editors, *Formal Program Development*, volume 755 of *Lecture Notes in Computer Science*, pages 7–42. Springer Verlag, 1993.

[BM97] R. Bird and O. Moor. *The Algebra of Programming*. Series in Computer Science. Prentice Hall, 1997.

[BV97] Roland Backhouse and Richard Verhoeven. Math∫pad: A system for on-line preparation of mathematical documents. *Software–Concepts and Tools*, pages 80–89, 1997.

[BZW+09] Andrew Bragdon, Robert Zeleznik, Brian Williamson, Timothy Miller, and Joseph J. LaViola, Jr. GestureBar: improving the approachability of gesture-based interfaces. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 2269–2278, New York, NY, USA, 2009. ACM.

[cit12] Caltech interface tools. Available from the author's website. URL: `http://www.ics.uci.edu/~arvo/software.html`, 2012. [Last accessed: 15 April 2012].

[Cro08] Jamie Cromack. Technology and learning-centered education: Research-based support for how the Tablet PC embodies the seven principles of good practice in undergraduate education. In *38th ASEE/IEEE Frontiers in Education Conference*. IEEE, 2008.

[cyg12] Cygwin (website). URL: `http://www.cygwin.com`, 2012. [Last accessed: 15 April 2012].

[det12] Detexify - latex symbol classifier (website). URL: `http://detexify.kirelabs.org/classify.html`, 2012. [Last accessed: 15 April 2012].

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[DS90] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer Verlag, NY, 1990.

[Fer10] João F. Ferreira. *Principles and Applications of Algorithmic Problem Solving*. PhD thesis, School of Computer Science, University of Nottingham, Nottingham, UK, November 2010.

[ffe05] The freehand formula entry system (website). URL: `http://www.cs.queensu.ca/drl/ffes/`, 2005. [Last accessed: 15 April 2012].

[Fit04] Thomas J. Fitzgerald. The Tablet PC takes its place in the classroom. New York Times. URL: `http://www.nytimes.com/2004/09/09/technology/circuits/09jott.html`, 2004. [Last accessed: 15 April 2012].

[FKS03] Mitsushi Fujimoto, Toshihiro Kanahori, and Masakazu Suzuki. Infty editor – a mathematics typesetting tool with a handwriting interface and graphical front-end to OpenXM servers. Available from `www.inftyproject.org/files/2003_RIMS_Fujimoto.pdf`, 2003.

[FM09] João F. Ferreira and Alexandra Mendes. Student's feedback on teaching mathematics through the calculational method. In *39th ASEE/IEEE Frontiers in Education Conference*. IEEE, 2009.

[FMBB09] João F. Ferreira, Alexandra Mendes, Roland Backhouse, and Luís S. Barbosa. Which mathematics for the information society? In *Teaching Formal Methods*, volume 5846 of *Lecture Notes in Computer Science*, pages 39–56. 2009.

[FMC$^+$11] João F. Ferreira, Alexandra Mendes, Alcino Cunha, Carlos Baquero, Paulo Silva, Luís Soares Barbosa, and J. N. Oliveira. Logic training through algorithmic problem solving. In *Third International Congress on Tools for Teaching Logic (TICTTL)*, volume 6680 of *LNAI*, pages 62–69. Springer Verlag, 2011.

[GFvGM90] David Gries, W.H.J. Feijen, A.J.M. van Gasteren, and J. Misra. *Beauty is our Business*. Springer Verlag, New York, 1990.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1994.

[GS93] David Gries and F. Schneider. *A Logical Approach to Discrete Mathematics*. Springer Verlag, New York, 1993.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM, 1984.

[Has08] Anne Mette Jonassen Hass. *Guide to Advanced Software Testing*. Artech House, 2008.

[Htw07] Swe Myo Htwe. *Lexicon Organisation and Contextual Methods For Online Handwritten Pitman's Shorthand Recognition*. PhD thesis, School of Computer Science, University of Nottingham, Nottingham, UK, 2007.

[HYP$^+$10] Ken Hinckley, Koji Yatani, Michel Pahud, Nicole Coddington, Jenny Rodenhouse, Andy Wilson, Hrvoje Benko, and Bill Buxton. Manual deskterity: an exploration of simultaneous pen + touch direct input. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, CHI EA '10, pages 2793–2802, New York, NY, USA, 2010. ACM.

[ieu12] Infty editor (website). `http://www.inftyproject.org/en/software.html`, 2012. [Last accessed: 15 April 2012].

[ink08] InkML toolkit (InkMLTk) project (website). URL: `http://inkmltk.sourceforge.net`, 2008. [Last accessed: 15 April 2012].

[ink11] InkML: Ink markup language, W3C recommendation (website). URL: `http://www.w3.org/TR/InkML`, 2011. [Last accessed: 15 April 2012].

[jim12] Java interactive mathematical handwriting recognizer (website). URL: `http://jimhr.sourceforge.net/`, 2012. [Last accessed: 15 April 2012].

[jmn11] JMathNotes (website). URL: `http://page.mi.fu-berlin.de/tapia/projects.php`, 2011. [Last accessed: 15 April 2012].

[JS03] Rob Jarrett and Philip Su. *Building Tablet PC Applications*. Microsoft Press, 2003.

[KDS06] Levent Burak Kara, Chris M. D'Eramo, and Kenji Shimada. Pen-based styling design of 3D geometry using concept sketches and template models. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, SPM '06, pages 149–160, New York, NY, USA, 2006. ACM.

[LLM$^+$08] George Labahn, Edward Lank, Scott MacLean, Mirette Marzouk, and David Tausky. Mathbrush: A system for doing math on pen-based devices. *Document Analysis Systems, IAPR International Workshop on*, 0:599–606, 2008.

[LMZL08] Chuajun Li, Timothy Miller, Robert Zeleznik, and Joseph LaViola. AlgoSketch: Algorithm sketching and interactive computation. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling 2008*, pages 175–182, 2008.

[LNH+01] James Lin, Mark W. Newman, Jason I. Hong, James A. Landay, and James A. L. Denim: An informal tool for early stage web site design. In *in Extended Abstracts of Human Factors in Computing Systems: CHI 2001*, pages 205–206, 2001.

[LREND10] Marcus Liwicki, Oleg Rostanin, Saher Mohamed El-Neklawy, and Andreas Dengel. Touch & write: a multi-touch table with pen-input. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, DAS '10, pages 479–484, New York, NY, USA, 2010. ACM.

[LWD08] Jie Li, Danli Wang, and Guozhong Dai. A pen-based 3D role modeling tool for children. In Zhigeng Pan, Adrian Cheok, Wolfgang Müller, and Abdennour El Rhalibi, editors, *Transactions on Edutainment I*, volume 5080 of *Lecture Notes in Computer Science*, pages 62–73. Springer Verlag, 2008.

[LZ04] Joseph J. LaViola, Jr. and Robert C. Zeleznik. Mathpad2: a system for the creation and exploration of mathematical sketches. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 432–440, New York, NY, USA, 2004. ACM.

[map12] Maplesoft - technical computing software for engineers, mathematicians, scientists, instructors and students (website). URL: `http://www.maplesoft.com`, 2012. [Last accessed: 15 April 2012].

[Mat99] Nicholas Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1999.

[mat06] MathPad^2: A system for the creation and exploration of mathematical sketches (website). URL: `http://www.cs.brown.edu/people/jjl/mathpad/`, 2006. [Last accessed: 15 April 2012].

[mat09] MathIS: Reinvigorating mathematics for the information society (website). URL : `http://twiki.di.uminho.pt/twiki/bin/view/Research/Matisse/MathIS`, 2009. [Last accessed: 15 April 2012].

[mat12] Mathpaper project (website). URL: `http://pen.cs.brown.edu/research.html#mathpaper`, 2012. [Last accessed: 15 April 2012].

[mbu12] Mathbrush: A penmath system (website). URL: `https://www.scg.uwaterloo.ca/mathbrush/index.php`, 2012. [Last accessed: 15 April 2012].

[Men08] Alexandra Mendes. Work in progress: Structure editing of handwritten mathematics. In *38th ASEE/IEEE Frontiers in Education Conference*. IEEE, 2008.

[Mic12] Microsoft Surface. URL: `http://www.microsoft.com/surface`, 2012. [Last accessed: 15 April 2012].

[mju12] Mathjournal, interactive journal for math (website). URL: `http://www.xthink.com/MathJournal.html`, 2012. [Last accessed: 15 April 2012].

[MK04] Dirk Materlik and Ulrich Kortenkamp. Pen-based input of geometric constructions. In *Mathematical User-Interfaces Workshop*, 2004.

[mm12] MathMagic, the ultimate equation editor on the planet (website). `http://www.mathmagic.com`, 2012. [Last accessed: 15 April 2012].

[mob12] Mobomath, the handwritten equation editor for calculations and technical documents (website). URL: `http://www.enventra.com/products/mobomath/overview.htm`, 2012. [Last accessed: 15 April 2012].

[msw12] Math speak & write (website). URL: `http://www.cs.berkeley.edu/~fateman/msw/msw.html`, 2012. [Last accessed: 15 April 2012].

[mt12] Mathtype – professional equation editor (website). URL: `http://www.chartwellyorke.com/mathtype.html`, 2012. [Last accessed: 15 April 2012].

[Mur98] Kevin Murphy. A brief introduction to graphical models and bayesian networks (website). URL: `http://www.cs.ubc.ca/~murphyk/Bayes/bayes.html`, 1998. [Last accessed: 15 April 2012].

[Nam05] Tek-Jin Nam. Sketch-based rapid prototyping platform for hardware-software integrated interactive products. In *CHI '05 extended abstracts on Human factors in computing systems*, CHI '05, pages 1689–1692, New York, NY, USA, 2005. ACM.

[nlu03] Natural log: Recognizing handwritten mathematics (website). URL: `http://www.ai.mit.edu/projects/natural-log`, 2003. [Last accessed: 15 April 2012].

[Oli05] Werner Oliver. Teaching mathematics: Tablet PC technology adds a new dimension. URL: `http://math.unipa.it/~grim/21_project/21_malasya_Olivier176-181_05.pdf`, 2005. Presented at Reform, Revolution and Paradigm Shifts in Mathematics Education. The Mathematics Education into the 21st Century Project, Universiti Teknologi Malaysia.

[Pea85] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, August 1985.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[Pol57] George Polya. *How to solve it*. Princeton University Press, 1957.

[ros12] Rose trees – wikipedia entry (website). URL: `http://en.wikipedia.org/wiki/Rose_Tree`, 2012. [Last accessed: 15 April 2012].

[rtr12] R-trees – wikipedia entry (website). URL: `http://en.wikipedia.org/wiki/R-tree`, 2012. [Last accessed: 15 April 2012].

[Rub94] Jeffrey Rubin. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley and Sons, Inc., 1994.

[SAHS04] Beth Simon, Ruth Anderson, Crystal Hoyer, and Jonathan Su. Preliminary experiences with a tablet pc based system to support active learning in computer science courses. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '04, pages 213–217, New York, NY, USA, 2004. ACM.

[sf12] C# Spatial Index (RTree) Library (website). URL: `http://sourceforge.net/projects/cspatialindexrt`, 2012. [Last accessed: 15 April 2012].

[sig05] Simple gesture recognition library (website). URL: `http://msdn.microsoft.com/en-us/library/aa480673.aspx`, 2005. [Last accessed: 15 April 2012].

[Smi99] Steve Smithies. Freehand formula entry system. Master's thesis, Computer Science, University of Otago, Dunedin, New Zealand, May 1999. Available from `http://research.cs.queensu.ca/drl/ffes`.

[sta12] starPad SDK (website). URL: `http://pen.cs.brown.edu/starpad.html`, 2012. [Last accessed: 15 April 2012].

[SW06] Elena Smirnova and Stephen Watt. A pen-based mathematical environment: Mathink. Technical Report TR-06-04, University of Western Ontario, 2006.

[sw10] Scientific WorkPlace: the integration of LaTeX typesetting and computer algebra (website). URL: `http://www.sciword.demon.co.uk/scientificworkplace.htm`, 2010. [Last accessed: 15 April 2012].

[tab12] Microsoft Tablet PC SDK v1.7. `http://msdn.microsoft.com/en-us/library/ms840460.aspx`, 2012. [Last accessed: 15 April 2012].

[VB99] Richard Verhoeven and Roland Backhouse. Towards tool support for program verification and construction. In Jeanette Wing, Jim Woodcock, and Jim Davies, editors, *FM '99 – Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1128–1146, September 1999.

[Ver00] Richard Verhoeven. *The Design of the Math∫pad Editor*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2000.

[vG90] A. J. M. van Gasteren. *On the Shape of Mathematical Arguments*, volume 445 of *Lecture Notes in Computer Science*. Springer Verlag, 1990.

[vim12] Vim, the editor. URL: `http://www.vim.org/`, 2012. [Last accessed: 15 April 2012].

[vis12] Microsoft Visual Studio (website). URL: `http://www.microsoft.com/visualstudio`, 2012. [Last accessed: 15 April 2012].

[WD03] Lothar Wenzel and Holger Dillner. Mathjournal - an interactive tool for the Tablet PC. Available online at `http://www.xthink.com/downloads/MathJournal_2003.pdf`, 2003.

[Zei99] Paul Zeitz. *The Art and Craft of Problem Solving*. John Wiley and Sons, Inc., 1999.

[ZMLL08] Robert Zeleznik, Timothy Miller, Chuanjun Li, and Joseph J. Laviola, Jr. MathPaper: Mathematical sketching with fluid support for interactive computation. In *SG '08: Proceedings of the 9th international symposium on Smart Graphics*, pages 20–32, Berlin, Heidelberg, 2008. Springer Verlag.

Teachers' Guide

Task Guide

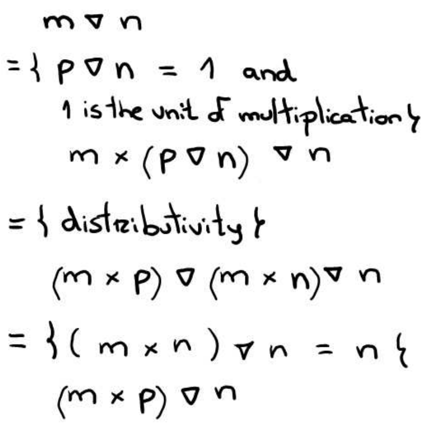
Task 1: Writing a Proof

In this task, you will prove the following theorem:

$$(m \times p)\nabla n = m\nabla n \ \Leftarrow\ p\nabla n = 1$$

The proof will start with the expression $m\nabla n$ and, assuming the right-hand side of the implication, the expression will be transformed into $(m \times p)\nabla n$. Please use the calculational proof format as shown in the images below (the hints should be ommited).

The complete proof is the following:



$m \nabla n$

$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$m \times (p \nabla n) \nabla n$

$= \{$ distributivity $\}$

$(m \times p) \nabla (m \times n) \nabla n$

$= \{ (m \times n) \nabla n = n \}$

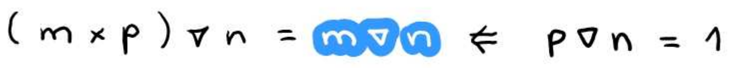
$(m \times p) \nabla n$

This task consists in going through all the steps of this proof. For each step, the level of difficulty of the actions performed is asked. Please use the scale shown to record the difficulty you have experienced (0 corresponds to "Very difficult" whilst 4 corresponds to "Very easy").
To handwrite (for instance, the equals sign) the **Pen mode** should be used. Click the button:

For all the other actions, use the **Lasso mode**. Click the button:

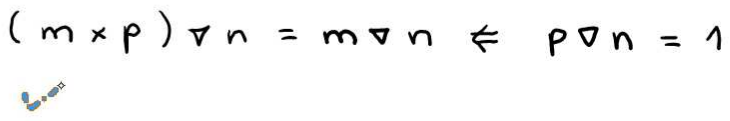
Step 1
Using the handwritten formula on the screen, please copy $m\nabla n$ to some location on the slide.

Start by selecting $m\nabla n$ by making a double tap on ∇. You should get the following result:



$$(m \times p)\nabla n = m\nabla n \Leftarrow p\nabla n = 1$$

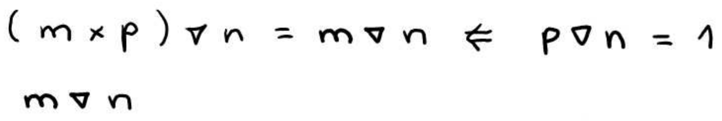
To perform the copy, use the gesture *Check* as in the following image:



$$(m \times p)\nabla n = m\nabla n \Leftarrow p\nabla n = 1$$

Please try to place the copy to the left so that enough space is left to expand the formula.

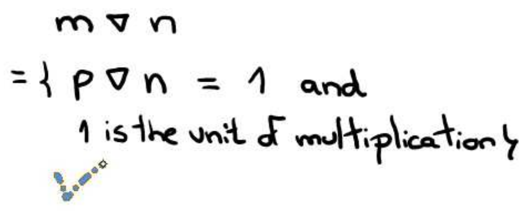
The result should be:



$$(m \times p)\nabla n = m\nabla n \Leftarrow p\nabla n = 1$$
$$m\nabla n$$

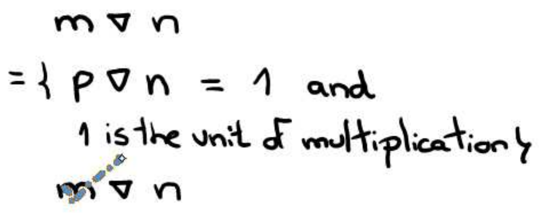
Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Copy	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 2
Copy the initial expression $m\nabla n$: select the expression by making a double tap over ∇ and use the gesture *Check* to copy the expression, as shown in the image:

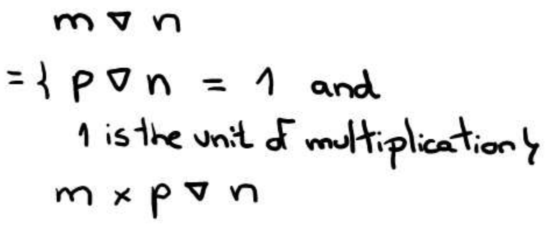


Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Copy	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 3
For the new copy, *Leibniz* is used to replace m by $m \times p$. For that, select $m \times p$ (again, double tap over $\times$) and use the *Check* gesture to substitute m by $m \times p$. The first point of the *Check* gesture has to be written over the m (as shown below):



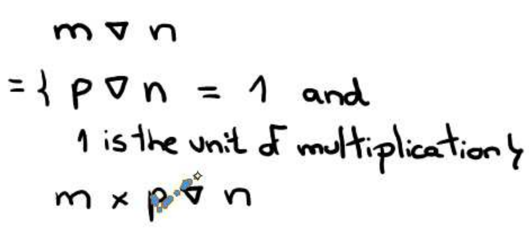
The result should be:



Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Leibniz	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 4

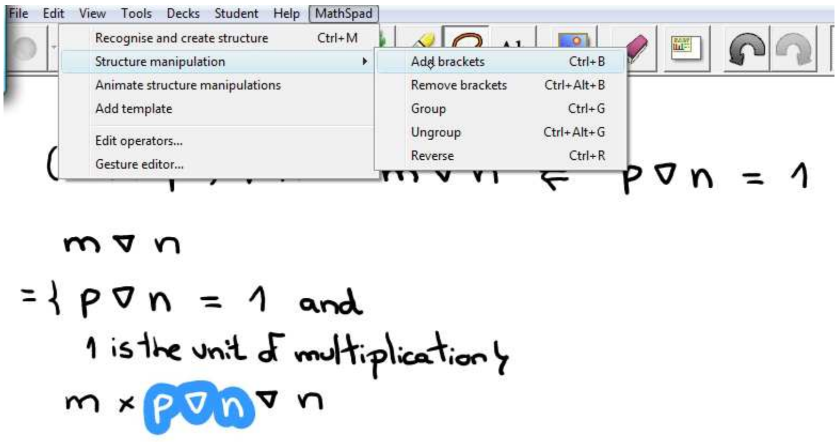
The next step is to replace p by $p\nabla n$. For that, select $p\nabla n$ (again, double tap over ∇) and use the *Check* gesture to substitute p by $p\nabla n$ (*Leibniz*). Please remember that the first point of the *Check* gesture has to be written over the p (as shown below):



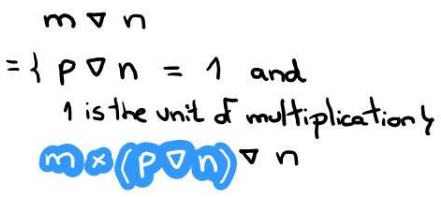
Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Leibniz	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 5

Add brackets to $p\nabla n$: first select the expression (double tap over ∇) then click on the menu *MathSpad* → *Structure manipulation* → *Add brackets*.



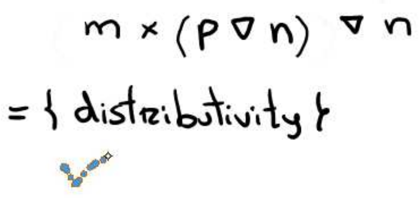
The result should be:



Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Add brackets	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 6

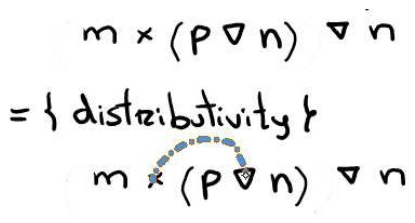
The resulting expression is copied: double tap over the last ∇ then using the gesture *Check* it is copied to the next step of the calculation as shown in the image:



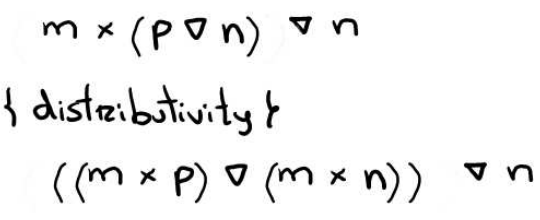
Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Copy	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 7

Apply distributivity of × over ∇ using a semi-circle starting over the × operator and ending over the first ∇, as shown in the image:



The result should be the following:

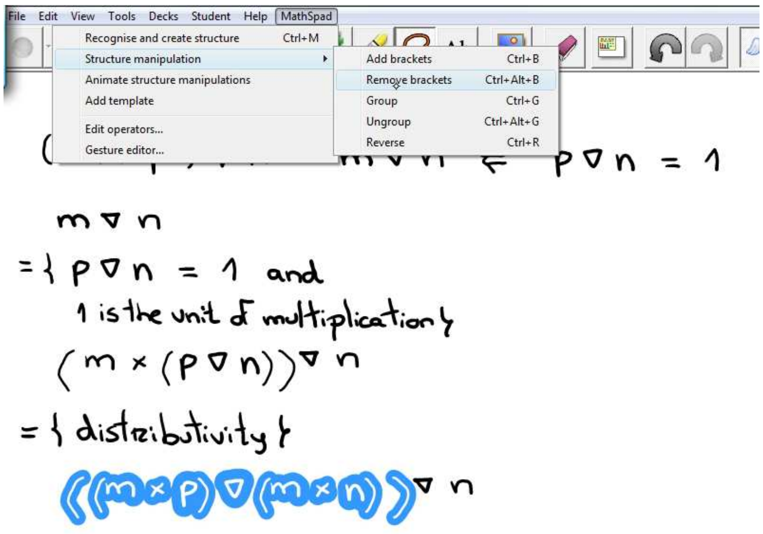


$$m \times (p \nabla n) \nabla n$$
$$= \{ \text{distributivity} \}$$
$$((m \times p) \nabla (m \times n)) \nabla n$$

Distributivity Very difficult 0 — 1 — 2 — 3 — 4 Very easy

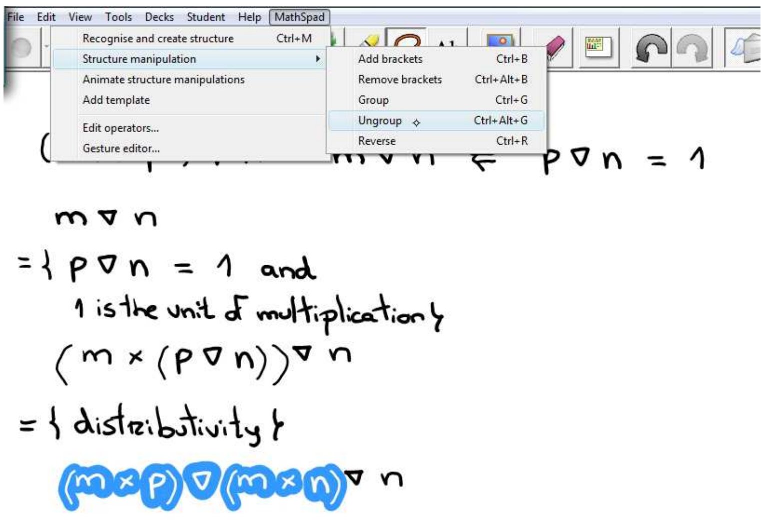
Step 8

Remove the brackets surrounding the expression $(m \times p) \nabla (m \times n)$. Select the expression including the external brackets (double tap on the first bracket) and click on the menu *MathSpad* → *Structure manipulation* → *Remove brackets*.



$$p \nabla n = 1$$
$$m \nabla n$$
$$= \{ p \nabla n = 1 \text{ and } 1 \text{ is the unit of multiplication} \}$$
$$(m \times (p \nabla n)) \nabla n$$
$$= \{ \text{distributivity} \}$$
$$((m \times p) \nabla (m \times n)) \nabla n$$

After removing the external brackets, it is not possible to select the subexpression $(m \times n) \nabla n$, because the expression $(m \times p) \nabla (m \times n)$ remains grouped. To proceed with the calculation, this expression should be ungrouped. For that, select the expression (double tap over ∇) and click on the menu *MathSpad* → *Structure manipulation* → *Ungroup*.


File Edit View Tools Decks Student Help MathSpad
Recognise and create structure Ctrl+M
Structure manipulation
Animate structure manipulations
Add template
Edit operators...
Gesture editor...
Add brackets Ctrl+B
Remove brackets Ctrl+Alt+B
Group Ctrl+G
Ungroup Ctrl+Alt+G
Reverse Ctrl+R

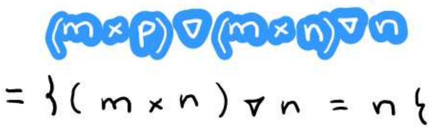
$$p \nabla n = 1$$
$$m \nabla n$$
$$= \{ p \nabla n = 1 \text{ and } 1 \text{ is the unit of multiplication} \}$$
$$(m \times (p \nabla n)) \nabla n$$
$$= \{ \text{distributivity} \}$$
$$(m \times p) \nabla (m \times n) \nabla n$$

Selection Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Removing brackets Very difficult 0 — 1 — 2 — 3 — 4 Very easy

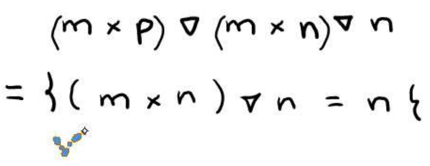
Ungroup Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 9
Double tap over ∇ twice (to expand the selection):



$$(m \times p) \nabla (m \times n) \nabla n$$
$$= \{ (m \times n) \nabla n = n \}$$

then, copy using the check gesture:



$$(m \times p) \nabla (m \times n) \nabla n$$
$$= \{ (m \times n) \nabla n = n \}$$

Assuming that $(m \times n) \nabla n = n$, replace $(m \times n) \nabla n$ by n. Select one n (double tap over n); use *Leibniz* to substitute (gesture *Check* over the last ∇).

$$(m \times p) \nabla (m \times n) \nabla n$$
$$= \}(m \times n) \nabla n = n\{$$
$$(m \times p) \nabla (m \times n) \nabla n$$

Selection	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Copy	Very difficult 0 — 1 — 2 — 3 — 4 Very easy
Leibniz	Very difficult 0 — 1 — 2 — 3 — 4 Very easy

Step 10

Save all your work. Click on *File* → *Save Deck As* and name the file task1.cp3.

Animation

This task consists of applying distributivity to an expression with the animations feature on. Please go to slide 2. Go to *MathSpad* → *Animate structure manipulations* to turn the animations on. Now, on the expression in slide 2, apply distributivity of $\times$ over ∇ with a semi-circle from $\times$ to ∇.

Task 3: Handwritten Sample

This task consists of collecting your handwritten representation of some mathematical texts. Please create a blank deck (go to *Decks* → *Create a Blank Whiteboard Deck*). Making sure that you are in the *Pen Mode*, write the mathematical text below on the blank deck. When you are finished, please save the deck as sample.cp3 (*File* → *Save Deck As*).

```
$\{M > 0 \ \wedge \ N > 0\}$
$m, n := M, N;$
$\{m \ \mathbf{gcd} \ n = M \ \mathbf{gcd} \ N\}$
$do$
  $n < m \ \rightarrow \ m := m - n$
$\Box$
  $m < n \ \rightarrow \ n := n - m$
$od$
$\{m = n = m \ \mathbf{gcd} \ n\}$
```

$$\begin{array}{ll} & A \\ = & \{ \quad \text{p} \quad \} \\ & B \\ \Leftarrow & \{ \quad \text{q} \quad \} \\ & C \end{array}$$

Thank you for all your hard work!

Appendix B

Teachers' Questionnaire

Questionnaire

In this test you will be given an exercise to complete using the software provided and a questionnaire to record your opinion about the software. The task consists of a set of predefined steps that will be given to you together with this questionnaire. The aim of the test is to assess whether the features you have used are useful for teaching courses that use the calculational method. We would appreciate if you could give us your sincere opinion through this anonymous questionnaire. The information provided will be used for the evaluation of the functionalities that you have used today. Thank you for your participation!

About you

I am a: ☐ Lecturer ☐ PhD Student
☐ Other. Please detail: ____________

About the method

Have you been exposed to the calculational method before?
☐ Yes ☐ No

Do you like the calculational method?
☐ Yes ☐ Neutral ☐ No

Do you have any experience with teaching using the calculational method?
☐ Yes ☐ No

About the software

In your opinion, how easy is it to complete the task using this software?
☐ Very easy ☐ Easy ☐ Neutral ☐ Difficult ☐ Extremely difficult

The features provided by this software can help with teaching mathematics.
☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

I prefer to manipulate a formula through the structure than manually moving the ink around to obtain the desired result.
☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

In task 2, you have seen an animation of the distributivity rule. Do you think animations would be useful for teaching?
☐ Yes ☐ No

Do you think that play/pause features for the animations would be useful?
☐ Yes ☐ No

Gestures are a good way to trigger actions.
☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

What did you like the most about the software?

____

What did you dislike the most about the software?

____

Did you have difficulties in using this software? If your answer is yes, please give details.

____

____

Do you have any suggestions on how to improve this software? If your answer is yes, please give details.

____

____

____

____

About Tablets

I prefer to use the handwritten representation of formulae than their representation in typeset.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

I find it easier to handwrite mathematics than writing it using a keyboard.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

Input using a pen is more precise than using the fingers.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

Thank you!

Appendix C

Students' Questionnaire

Questionnaire

The aim of today's test is to assess whether the features that are going to be presented to you today are useful for teaching courses that use the calculational method. We would appreciate if you could give us your sincere opinion through this anonymous questionnaire. The information provided will be used for the evaluation of the functionalities that were presented to you today. Thank you for your participation!

About you

Age ___

Degree ☐ LEI ☐ LCC

About the method

Have you been exposed to the calculational method before?
☐ Yes ☐ No

Do you like the calculational method?
☐ Yes ☐ Neutral ☐ No

About the software

The example shown to you is based on symbolic manipulation. Did the software help you understanding the manipulations involved?
□ Yes □ No

This software can help me doing calculations by myself.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

This software can improve my experience in the classroom.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

This software can help me with learning calculational mathematics.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

This software can help me with learning general mathematics.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

Being able to send answers from my computer to the teacher's makes it easier to participate in the lectures.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

Being able to send answers from my computer to the teacher's makes the lectures more interactive.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

In the demo you were shown an animation. Do you think animations would be useful to understand better how algebraic rules are applied?
□ Yes □ No

Do you think that play/pause features for the animations would be useful?
□ Yes □ No

Gestures are a good way to trigger actions.
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

The gestures used during the presentation are adequate to the operations to

which they are associated.

□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

If this software became available during your attendance to the "Program Calculus" course, would you have used it?

□ Yes □ No

Would you use this software for anything else?

□ Yes. What for? ______________________________

□ No

What did you like the most about the software presented?

____

What did you dislike the most about the software presented?

____

Do you have any suggestions on how to improve this software? If your answer is yes, please give details.

____

____

____

____

General

I find it easier to handwrite mathematics than writing it using a keyboard

□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

Input using a pen is more precise than using the fingers.

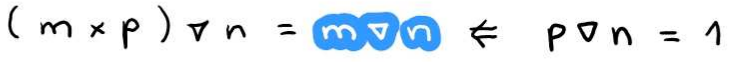
□ Strongly Agree □ Agree □ Neutral □ Disagree □ Strongly Disagree

Thank you!

APPENDIX D

Using the current system to write a proof

In this section, a proof of the theorem

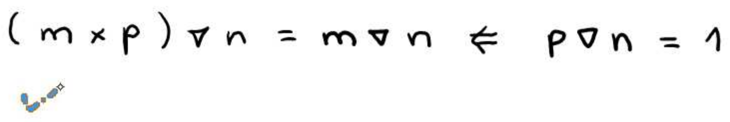


$$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$$

is presented to demonstrate how the current system can be used to do mathematics. As the library by itself cannot be used to demonstrate the features, the extended version of the *Classroom Presenter* described in chapter 5 is used.

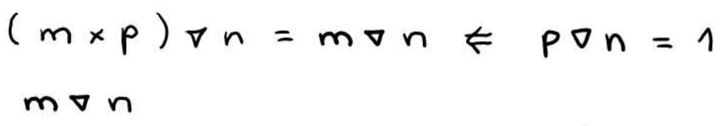
The example shows how the system can make explicit the dynamics of the symbols involved in mathematical calculations. The example is very detailed to facilitate the demonstration of the dynamics involved.

Proof:
The proof shown in [Fer10] starts with the expression $m \nabla n$ and, assuming the right-hand side of the implication, transforms it into $(m \times p) \nabla n$. To prove it using the tool, the teacher starts by writing the theorem and recognising it. The next step is to copy $m \nabla n$. For that, the teacher selects it in the theorem above and copies it using the gesture *Check*. The previous image shows the selection; the next image shows the writing of the gesture:



$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

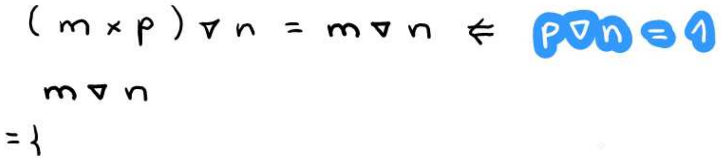
The result is:



$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

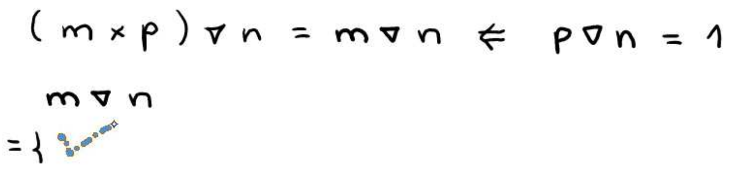
Next, the hint for the next step is written. For that, the teacher selects the expression $p\nabla n = 1$ and uses the *Check* gesture, once again, to copy:



$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

$= \{$

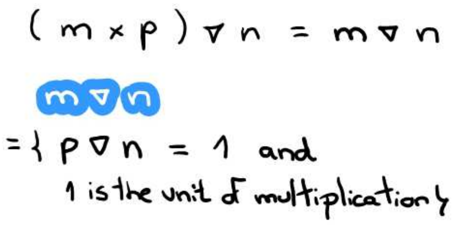


$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

$= \{$

Some textual justification is added as well.



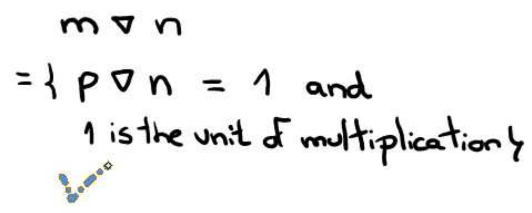
$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

$= \{ p \nabla n = 1$ and

1 is the unit of multiplication $\}$

In this example, the proof format is handwritten without structure. Templates for the proof format could be used, but currently writing text in the hints is not practical. Moreover, the example demonstrates that the system is flexible, i.e. the users are not forced to create structure for everything they write. Now, the initial expression $m\nabla n$ is selected and copied:

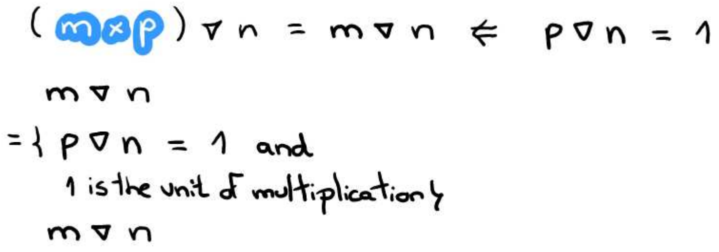


$m \nabla n$

$= \{ p \nabla n = 1$ and

1 is the unit of multiplication $\}$

Next, the new copy is manipulated. For that, *Leibniz* is used to replace m by $m \times p$. So, $m \times p$ is selected:



$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

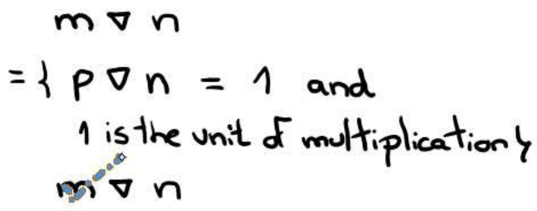
$m \nabla n$

$= \{ p \nabla n = 1$ and

1 is the unit of multiplication $\}$

$m \nabla n$

and *Leibniz* is applied to m:



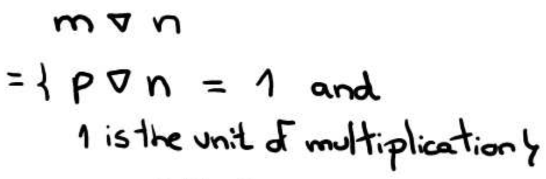
$m \nabla n$

$= \{ p \nabla n = 1$ and

1 is the unit of multiplication $\}$

$m \nabla n$

The result is:



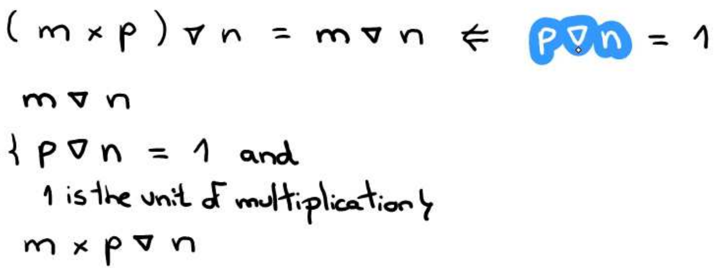
$m \nabla n$

$= \{ p \nabla n = 1$ and

1 is the unit of multiplication $\}$

$m \times p \nabla n$

The next step is to replace p by $p \nabla n$. The expression is selected:



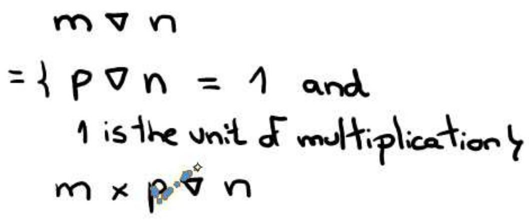
$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$m \times p \nabla n$

and *Leibniz* applied:

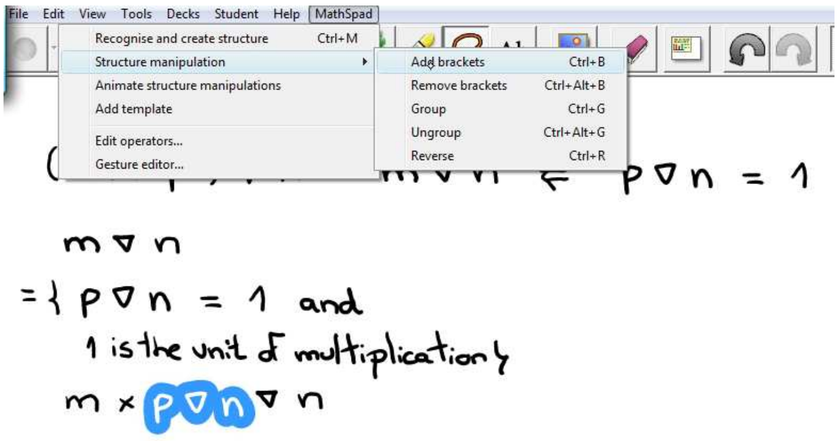


$m \nabla n$

$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$m \times p \nabla n$

Although the structure obtained reflects the correct precedence, the *Leibniz* rule does not add any brackets. At this stage, the teacher can add brackets to $p\nabla n$ to help the students read the expression:



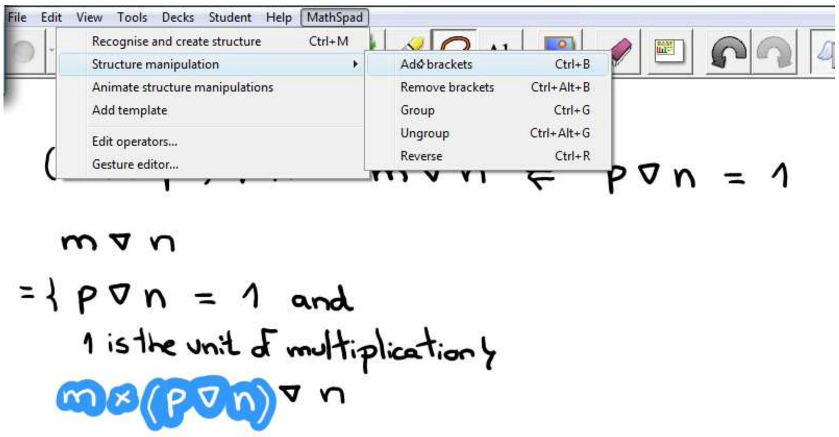
$\Leftarrow p \nabla n = 1$

$m \nabla n$

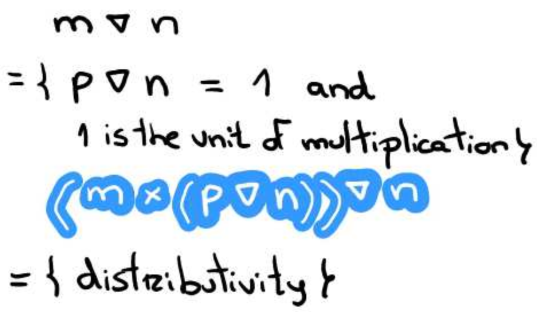
$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$m \times p \nabla n$

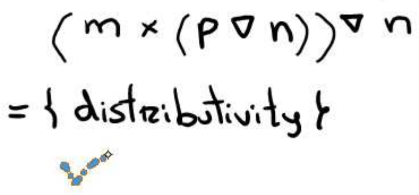
Brackets can also be added to $m \times (p \nabla n)$:



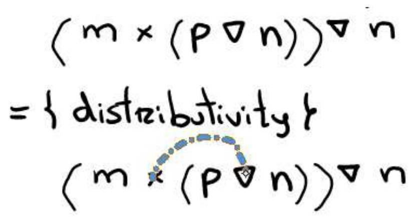
Next, the resulting expression is copied. First, the teacher selects it:



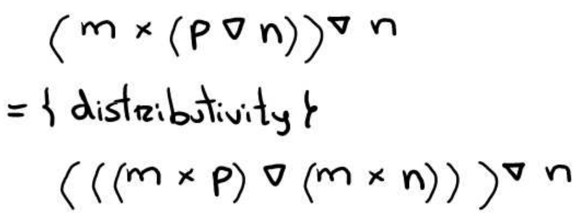
then copies it:



and, finally, distributivity is applied:



The result is the following:

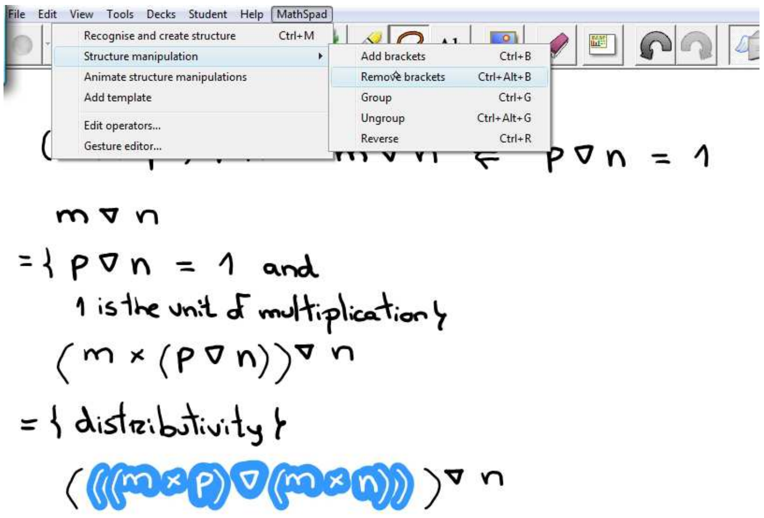


$(m \times (p \nabla n)) \nabla n$

$= \{$ distributivity $\}$

$(((m \times p) \nabla (m \times n))) \nabla n$

The brackets surrounding the expression $(m \times p) \nabla (m \times n)$ are removed in two steps:


File Edit View Tools Decks Student Help MathSpad
Recognise and create structure Ctrl+M
Structure manipulation
Animate structure manipulations
Add template
Edit operators...
Gesture editor...
Add brackets Ctrl+B
Remove brackets Ctrl+Alt+B
Group Ctrl+G
Ungroup Ctrl+Alt+G
Reverse Ctrl+R

$\Leftarrow p \nabla n = 1$

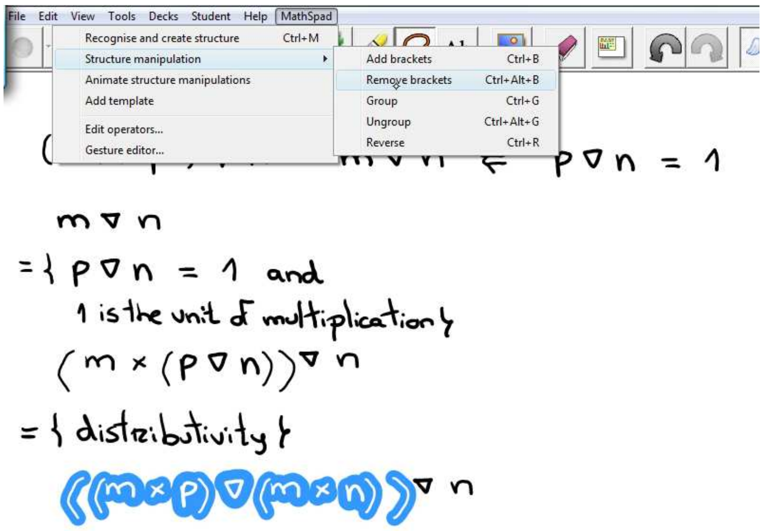
$m \nabla n$

$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$(m \times (p \nabla n)) \nabla n$

$= \{$ distributivity $\}$

$(((m \times p) \nabla (m \times n))) \nabla n$


File Edit View Tools Decks Student Help MathSpad
Recognise and create structure Ctrl+M
Structure manipulation
Animate structure manipulations
Add template
Edit operators...
Gesture editor...
Add brackets Ctrl+B
Remove brackets Ctrl+Alt+B
Group Ctrl+G
Ungroup Ctrl+Alt+G
Reverse Ctrl+R

$\Leftarrow p \nabla n = 1$

$m \nabla n$

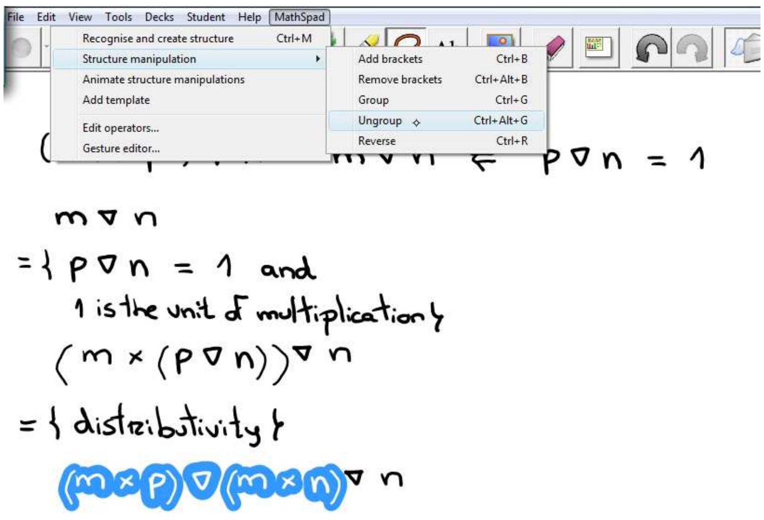
$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$(m \times (p \nabla n)) \nabla n$

$= \{$ distributivity $\}$

$((m \times p) \nabla (m \times n)) \nabla n$

After removing the two sets of brackets, it is not possible to select the subexpression $(m \times n) \nabla n$, because the expression $(m \times p) \nabla (m \times n)$ remains grouped. To proceed with the calculation, the teacher has to ungroup it:


File Edit View Tools Decks Student Help MathSpad
Recognise and create structure Ctrl+M
Structure manipulation
Animate structure manipulations
Add template
Edit operators...
Gesture editor...
Add brackets Ctrl+B
Remove brackets Ctrl+Alt+B
Group Ctrl+G
Ungroup Ctrl+Alt+G
Reverse Ctrl+R

$m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

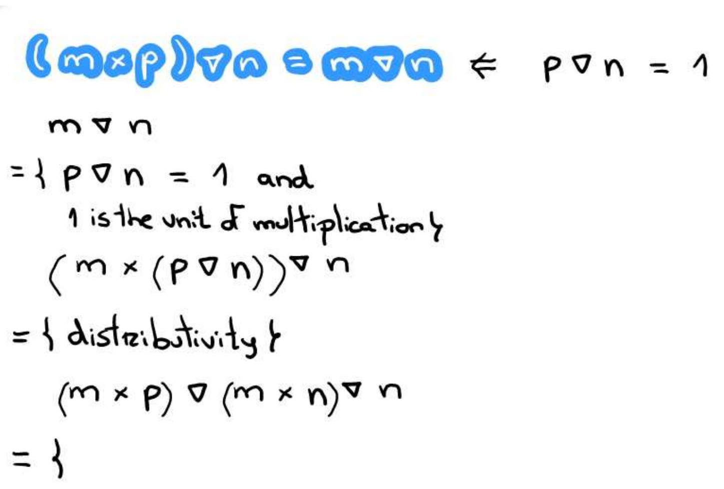
$= \{$ $p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$(m \times (p \nabla n)) \nabla n$

$= \{$ distributivity $\}$

$(m \times p) \nabla (m \times n) \nabla n$

After the ungroup, the three operands of ∇ are in the same level of the expression's tree. Now, the following step consists of replacing $(m \times n)\nabla n$ by n. A way to write the hint with structure consists in copying and changing the expression $(m \times p)\nabla n = m\nabla n$.First, it is selected:



$(m \times p) \nabla n = m \nabla n \Leftarrow p \nabla n = 1$

$m \nabla n$

$= \{$ $p \nabla n = 1$ and 1 is the unit of multiplication $\}$

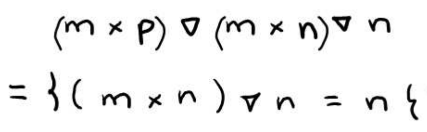
$(m \times (p \nabla n)) \nabla n$

$= \{$ distributivity $\}$

$(m \times p) \nabla (m \times n) \nabla n$

$= \{$

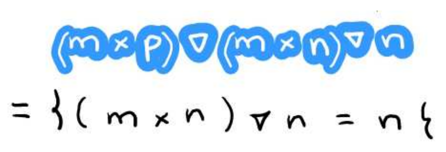
then, it is copied to the hint and manipulated to obtain the following (the steps of this manipulation were omitted, as they consist of simple substitutions):



$(m \times p) \nabla (m \times n) \nabla n$

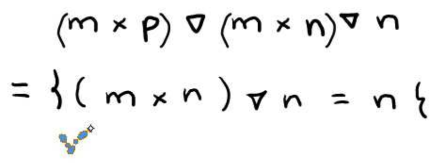
$= \{ (m \times n) \nabla n = n \}$

Next, the expression that was obtained in the previous step of the calculation, is selected:



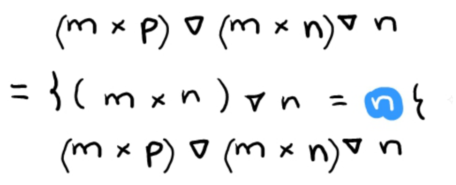
$$(m\times p)\nabla(m\times n)\nabla n$$
$$=\}(m\times n)\nabla n = n\{$$

then, copied:



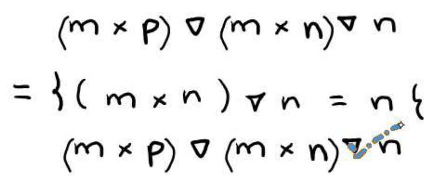
$$(m\times p)\nabla(m\times n)\nabla n$$
$$=\}(m\times n)\nabla n = n\{$$

and, finally, the hint is used to replace $(m\times n)\nabla n$ by n:



$$(m\times p)\nabla(m\times n)\nabla n$$
$$=\}(m\times n)\nabla n = n\{$$
$$(m\times p)\nabla(m\times n)\nabla n$$

Using *Leibniz*:



$$(m\times p)\nabla(m\times n)\nabla n$$
$$=\}(m\times n)\nabla n = n\{$$
$$(m\times p)\nabla(m\times n)\nabla n$$

The final result is obtained and the proof is concluded. The complete proof is as follows:

$m \nabla n$

$= \{ p \nabla n = 1$ and 1 is the unit of multiplication $\}$

$(m \times (p \nabla n)) \nabla n$

$= \{$ distributivity $\}$

$(m \times p) \nabla (m \times n) \nabla n$

$= \{ (m \times n) \nabla n = n \}$

$(m \times p) \nabla n$

Although the brackets in this example could benefit from a better normalisation, the author believes that this proof illustrates well the advantages of having a structure editor of handwritten mathematics for teaching. Not only the steps are made easier, but also the teacher can be sure that manipulation errors will be avoided, since the tool will calculate the results of the manipulations for them.