Structure Editing of Handwritten Mathematics

Improving the Computer Support for the Calculational Method

Alexandra Mendes
Computer Science Department
York St John University, UK
a.mendes@yorksj.ac.uk

Roland Backhouse
School of Computer Science
University of Nottingham, UK
roland.backhouse@nottingham.ac.uk

João F. Ferreira
School of Computing
Teesside University, UK

HASLab/INESC TEC
Universidade do Minho, Portugal
joao@joaoff.com

ABSTRACT

We present a structure editor that aims to facilitate the presentation and manipulation of handwritten mathematical expressions. The editor is oriented to the calculational mathematics involved in algorithmic problem solving and it provides features that allow reliable structure manipulation of mathematical formulae, as well as flexible and interactive presentations. We describe some of its most important features, including the use of gestures to manipulate algebraic formulae, the structured selection of expressions, definition and redefinition of operators in runtime, gesture's editor, and handwritten templates. The editor is made available in the form of a C# class library which can be easily used to extend existing tools. For example, we have extended Classroom Presenter, a tool for ink-based teaching presentations and classroom interaction. We have tested and evaluated the editor with target users. The results obtained seem to indicate that the software is usable, suitable for its purpose and a valuable contribution to teaching and learning algorithmic problem solving.

Author Keywords

handwritten mathematics; structure editor; Tablet PCs; computer science education; mathematics education; calculational method; C# class library; gestures

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces

General Terms

Algorithms; Design; Mathematics; Reliability; Human Factors

INTRODUCTION

Students, scientists, and engineers have the need to effectively communicate and present mathematical results. For that, computers are often used, but writing and presenting mathematical content using a computer is hard. The difficulty in writing is mainly due to the input devices commonly used: 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. For presentations, the blackboard, transparencies, and electronic slides are commonly used, but these media have some limitations. For instance, although the use of the blackboard allows interaction with the audience, it requires the presenter to write everything during the presentation, which is time-consuming and can lead to errors. Ordinary transparencies and electronic slides can be prepared in advance, avoiding errors and saving time. However, they work more like a guideline than as a medium for discussion and interaction since it is difficult to adjust them during the presentation. This does not encourage interactivity and, in presentations involving mathematical content, there is often the need to provide examples and details according to queries and responses from the audience. Moreover, electronic slides are usually created using the mouse and the keyboard as input devices, and these make it hard and laborious to write documents containing mathematics.

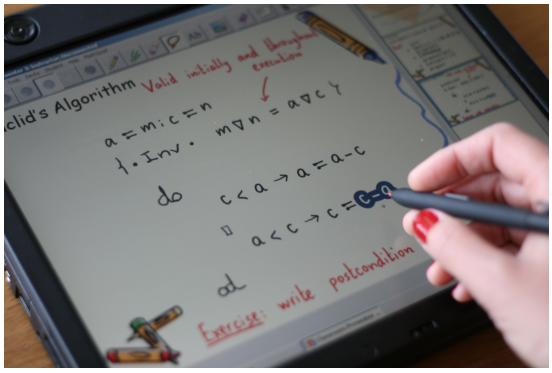


Figure 1. MST editor in action on a Tablet PC. Selecting well-formed sub-expressions is necessary to manipulate mathematical expressions.

An easier way to write and present mathematical content is to use pen-based devices like Tablet PCs and handwrite the desired mathematical content [8, 12, 19]. We currently use a Tablet PC to teach algorithmic problem solving to first-year undergraduate students and we find that it simplifies the writ-

Accepted version by ITS'14.

ing of mathematical content. Moreover, the feedback from the students shows that they enjoy the interactivity that comes with the use of the Tablet PC. However, because teaching algorithmic problem solving involves a large number of syntactic manipulations, we feel the need for a structure editor able to reliably manipulate the non-standard mathematics that we use.

In an attempt to combine the benefits of Tablet PCs with the advantages of using the computer's capabilities to manipulate structured information reliably, we have developed a structure editor[1] named Mathpad Tablet (MST). This editor aims to facilitate the presentation and manipulation of mathematical content by providing a way to input and manipulate structured handwritten mathematics, and by providing features that contribute to flexible and interactive presentations. Figure 1 shows the editor in action. As pictured, the tool allows the use of a stylus pen to handwrite and manipulate mathematical content (like algorithms). The main audience of the system are teachers presenting mathematical content in the calculational style that is used, for example, in algorithmic problem solving [2, 3, 10, 14, 22], but it can also be used in other contexts.

The novelty of the MST editor and what distinguishes it from existing related tools is that:

- it is the first structure editor for handwritten mathematics oriented to the calculational mathematics involved in algorithmic problem solving;
- it allows definition and redefinition of mathematical operators in runtime;
- it supports the definition of handwritten templates;
- it is developed as a C# library, which can be easily used to extend and combine existing tools.

It is important to note that the focus of this work is on structure editing and not on handwriting recognition. In this paper, we describe the context, goals, and design of the MST editor. We discuss the integration of the editor with an existing tool, Classroom Presenter [1], and we show how it can be used to do and teach mathematics. We show the results of an evaluation performed with target users/audience and we conclude with a discussion on the current issues of the system and how we plan to develop it further.

RELATED WORK

There exist several tools and packages that are capable of dealing with handwritten mathematics. For example, the *Freehand Formula Entry System* (FFES) [20] allows the freehand entry and editing of formulae using a pen and a tablet. FFES also allows the generation of LaTeX from its input. Another existing tool is the *Infty Editor* [21]. This editor supports 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, including LaTeX and MathML. These tools act mainly as mathematics input systems, while our tool aims to provide an editor for complete mathematical documents and presentations with the ability to manipulate the mathematical structure of the handwritten expressions.

More advanced tools are able to provide mathematical sketching, which is a pen-based approach for mathematical problem solving. Users write mathematics and supporting diagrams to create graphical animations that can be manipulated and can verify the correctness of the mathematics. The concept of mathematical sketching was first developed in the seminal work of *MathPad*^{2} [17]. Another example is *MathBrush* [16], which allows the handwritten input and recognition of mathematical expressions. It allows the use of gestures to manipulate expressions and is able to evaluate them by passing them to a computer algebra system. *MathPaper* [25] is another tool that recognizes 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* [18] 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. *MathJournal* [24] is an interactive tool for the Tablet PC that provides a natural and intuitive environment for mathematical and engineering problem solving. The software recognizes, interprets and provides solutions for handwritten or hand-drawn mathematical and engineering constructions. This tool recognizes a great number of mathematical symbols. It evaluates several kinds of mathematical expressions and, given an expression, plots the corresponding graph. Our work overlaps with the works described above in that it also supports handwritten input, manipulation of mathematical expressions, and the use of gestures. Some of the gestures available in the MST editor are similar to the gestures available in these tools (e.g. like in *MathPad*^{2}, the scratch-out gesture is used to erase content). However, the default actions that we associate with gestures are defined based on our teaching experience and on what we usually do on the blackboard (e.g. the semi-circle is used to apply distributivity). These associations can be changed by the user at any time using the gesture editor provided. It is important to note that our tool also differs from these works on the emphasis and domain of application: while these tools emphasize the recognition and evaluation of expressions, our focus is on syntactic manipulation of mathematical expressions and composition of mathematical structures, particularly applied to the calculational method supporting algorithmic problem solving.

Recent mathematical sketching tools that emphasize diagrammatic reasoning are *VectorPad* [5], which allows users to write down vector mathematics and present animations

[1]We use the terms *editor* and *library* interchangeably to refer to the work presented in this paper. Even though the work is available in the form of a library, its features form a structure editor.

illustrating different vector operations; *SetPad* [7], which allows users to write down set expressions and presents Venn diagrams associated with them; and *SketchSet* [23], a sketch tool for Euler diagrams whose curves can be arbitrary shapes. Finally, Hands-On Math [26] is a multi-touch and pen-based system (running on a Microsoft Surface) which allows freeform sketching and visualization of simple algebraic expressions. Similar to *SetPad* and *Hands-on Math*, we support the selection of sub-expressions and non-trivial manipulation rules like distributivity; however, a difference is that in our tool all the results of manipulations remain handwritten, i.e., there is no conversion to typeset. Also, to the best of our knowledge, none of the tools described in this section support user-definition of new mathematical operators in runtime nor the definition of handwritten templates.

In terms of mathematical structure editors designed to be used with keyboard and mouse, the closest work is Math∫pad [4]. In fact, the Math∫pad system has inspired the project presented in this paper. Its focus is on calculational mathematics and its ultimate goal 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 Math∫pad's creators believe the main problem lies with the interaction limitations of computers (see page 15 of [4]). One of the goals of the MST editor is to overcome these limitations.

CONTEXT AND GOALS

The MST editor was developed in the context of teaching and research on correct-by-construction program design. More than two decades of research in this area have created a new discipline of algorithmic problem solving and shed light on the underlying mathematical structures, modeling, and reasoning principles. Starting with the pioneering work of Dijkstra and Gries [10, 14], a *calculational method* emerged, emphasizing the use of systematic mathematical calculation in the design of algorithms. Exponents of the calculational method prefer to work with uninterpreted formulae and manipulate them according to their symbols and associated rules (the motto is: "*let the symbols do the work*").

An example of a representative calculational proof is Van Gasteren's proof [9] of the theorem

$$(1) \qquad (m{\times}p)\nabla n{=}m\nabla n \;\Leftarrow\; p\nabla n{=}1 \quad ,$$

where ∇ represents the greatest common divisor. The proof is as follows:

$$\begin{array}{ll} & m\nabla n \\ = & \quad \{ \quad p\nabla n = 1 \text{ and } 1 \text{ is the unit} \\ & \quad\quad\quad \text{of multiplication} \quad \} \\ & m{\times}(p\nabla n)\,\nabla\, n \\ = & \quad \{ \quad \text{distributivity and associativity} \quad \} \\ & (m{\times}p)\,\nabla\,(m{\times}n)\,\nabla\, n \\ = & \quad \{ \quad (m{\times}n)\nabla n = n \quad \} \\ & (m{\times}p)\nabla n \quad . \end{array}$$

$$\begin{array}{lcll} m & \nabla & n & \text{(initial expression)} \\ \boxed{m{\times}p} & \nabla & n & \\ m{\times}\boxed{(p\nabla n)} & \nabla & n & \text{(final expression)} \end{array}$$

Figure 2. Transformation of $m\nabla n$ into $m{\times}(p\nabla n)\nabla n$. The red sub-expressions are replaced by the boxed expressions.

Note how each step of a calculational proof is usually accompanied by a hint justifying the validity of that step. Some relevant advantages of this format are that the hints reduce the search space, there are no repeated intermediate expressions, and we can immediately conclude that $m\nabla n \;=\; (m{\times}p)\nabla n$, just by looking at the first and last expressions and at the relations connecting them. The equality holds if the properties used in the hints are valid, meaning that $p\nabla n \;=\; 1$ is a sufficient condition. This calculational format is used in several research communities (e.g. by researchers on functional programming) and in teaching at many universities (e.g. a study suggests that discrete maths students prefer and understand calculational proofs better [11]).

Handwritten presentations of calculational proofs

The motivation for the creation of the MST editor is the need for a software tool supporting the teaching of algorithmic problem solving, in particular, using the calculational style. Teaching algorithmic problem solving involves a great deal of syntactic manipulations of uninterpreted and unconventional mathematical formulae that existing tools do not support. Working with uninterpreted formulae and manipulating them in a syntactic way requires a great deal of copy and substitution of expressions. For instance, in a handwritten presentation of the proof shown above, we would have to handwrite the expression $p\nabla n$ three times (once in the theorem statement, once in the first hint, and once in the result of the first step). In general, expressions can be arbitrarily large, meaning that copying and substituting them by hand can easily result in errors. The emphasis on substitution of equals for equals leads to proofs where each step only changes part of an expression. For example, in the first step of the proof above, only the first operand of ∇ is changed; in the third step, only the second operand of the first ∇ is changed. Moreover, syntactic manipulations like those on the distributivity step require a substantial mental effort from the writer as they have to perform in their mind a sequence of steps and write down the final result — which again, can easily result in errors.

To improve the user interaction with handwritten calculational proofs, there is the need for a structure editor providing straightforward, flexible and reliable manipulation of mathematical handwritten formulae. The editor should assist with the process of doing mathematics and ensure that human errors are less likely to be introduced. For example, a reliable way to obtain the result of the first step of the proof above is to copy the expression $m\nabla n$, replace the first operand by the expression $m{\times}p$, and finally, replace the p in the newly introduced sub-expression by $p\nabla n$. Figure 2 illustrates the three steps involved in this transformation.

The MST editor supports the type of manipulations described above, but it also considers other aspects relevant to mathematical structure editors. For example, above we have chosen the symbol ∇ to represent the greatest common divisor, because we wanted to demonstrate a common need when doing mathematics: we often need to define and conjecture new operators that satisfy certain algebraic rules. This is supported in the MST editor because users can add and redefine operators as they wish. When using the MST editor, everything remains handwritten even after manipulations are performed, as it is likely that the user will further add handwritten text, handwritten mathematical expressions, and possibly unstructured handwritten blocks (like sketches and diagrams) to the document. It is undesirable to mix different writing and font styles, as doing so, can make presentations confusing. Furthermore, certain actions can be triggered by the use of gestures, helping to make writing mathematics a continuous process, without interruption from having to select options from menus.

In summary, the MST editor was created to overcome some common obstacles when using a calculational style to teach algorithmic problem solving: i) difficulty writing mathematics in a digital format; ii) errors introduced during presentations, e.g. whilst writing calculations; iii) demonstrate the dynamics of algorithmic problem solving — that is, work with uninterpreted formulae and manipulate them according to their symbols and associated rules; iv) keep the audience engaged. To the best of our knowledge, there is no other tool that supports calculational mathematics in an handwritten form with the facilities needed in a research and teaching environment.

The focus of this work is on structure editing and not on recognition of handwriting. As such, we do not discuss problems related with handwriting recognition. The editor is built assuming that the correct recognition result is obtained and it works independently of the recognition engine. As it is not realistic to assume that a recognizer will always return the correct result, the library includes a simple editor for the recognition results. This editor allows the user to correct any errors in the recognition results. The library can be used even without a recognizer: if the user handwrites an expression and manually inputs the correct recognition result, the expressions will have the correct structure. This means that all the features still work, since they only depend on the existing structure.

THE MST EDITOR

The MST editor assists in the presentation of the dynamics of algorithmic problem solving [2, 3, 22], by allowing straightforward, flexible and reliable manipulation of non-standard mathematical expressions [13, 14]. All the relevant features of the editor are implemented in the form of a class library that we call *MST library*. The central feature of the library 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, while avoiding the introduction of errors. To facilitate structured manipulation, we introduce gestures to apply manipulation rules. By default, and where applicable, the gestures used for each rule match the visual idea that is usually associated with its application: for instance, for distributivity we use a semi-circle starting at the operator that will be distributed and ending over the operator through which it will be distributed — this matches our visual idea that one operator will be propagated over the other. All default gestures were chosen based on our teaching experience and on what we usually do on the blackboard. However, the association between gestures and rules can be customized at anytime to suit the user's needs. As each user is different, fixed gestures may not be suitable for everyone.

In the context of teaching, applying an algebraic rule and obtaining the result instantaneously, may not be good enough. With this in mind, the library supports the animation of certain structure manipulations. The purpose of these animations is to show, step-by-step, how a rule is applied to an expression. We believe that this can help students understand how the rule works.

Other features of this library include an operator editor that allows the definition (in runtime) of operators, redefinition of their precedences and kinds (infix, prefix, etc.), user-defined handwritten representations of characters, handwritten templates and automatic space adjustment of some mathematical structures.

Our class library is implemented in C# (using Microsoft's Tablet PC API) and it was created in a way that makes it easy to be used and extended by other programmers[2]. In particular, it can evolve with new technological developments: for instance, it is possible to change the handwritten mathematics recognizer being used[3] — which is important, because the area of handwritten recognition is still evolving.

In the following sections we provide more details about the main features of the class library.

Mathematical Structure

Mathematical structure is very important for writing calculations. Without knowing the formulae's structure, we can not manipulate it. Selecting well-formed sub-expressions is necessary to enable manipulations. For that reason, we provide a reliable and simple way of selecting expressions and sub-expressions.

Using the MST library, one can click[4] in separate elements of an expression to obtain a selection according to its structure. To illustrate how this works, consider the expression $a + b \times c + d + e$. A single click in any of the symbols that form the expression, selects the symbol itself. A double-click, however, has a different effect. If one double-clicks in one of the variables (a, b, c or d) the selection obtained is still

[2]The library is organized in modules, each one supporting a main feature of the system. It also follows certain design patterns, for example the Strategy pattern for choosing the recognizer to be used.

[3]Currently, the system is based on a proprietary mathematical symbol recognizer from MapleSoft.

[4]We use the term *click* to denote the action of pressing down, and possibly releasing a button, on a pointing device. When using a Tablet PC and a pen, we can also use the term *tap*.

the character that was clicked — no other selection can be obtained by clicking on a variable. However, if a double-click is performed on an operation symbol, the selection expands. In Figure 3, the result of a double-click on the last addition operator is shown. Because the structure of the addition operator is known, it selects, together with the operation symbol, the two arguments of the operator (d and e).

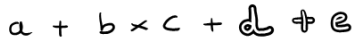


Figure 3. Double-click on the last addition operator.

Figures 4 and 5 show that a double-click on the first and on the second addition operators, respectively, selects 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, it is selected as one of the arguments of the addition. For the first addition symbol, if the selection was only $a + b$ it would not replicate the semantics of the original expression.

Figure 4. Double-click on the first addition operator.

Figure 5. Double-click on the second addition operator.

It is possible to extend the selection by continuously clicking in one of the operators. For instance, if, with $a + b{\times}c$ selected (as in Figure 4), one does another double-click on the first addition, the selection is extended to $a + b{\times}c + d$. Double-clicking on the first addition again selects the whole expression — that is, the selection is extended to contain $+e$.

One action that benefits heavily from this selection feature, is the copy of expressions and sub-expressions. One way to copy expressions using our library is to select the expression that one wants to copy and then, while holding the side button of the pen, touch with the tip of the pen in the area of the screen where the copy should be placed and release the button. Alternatively, a gesture can be used. Another action that depends on the accurate selection feature is the algebraic manipulation of expressions. In the next section, we give more details about this feature.

Structure Manipulation

One of the main goals of the library is to provide the means to perform syntactic manipulation of mathematical formulae. The idea is to apply rules to expressions automatically — automatically in the sense that the user does not have to determine the result of applying the rule. The main motivation behind the automatic use of rules is the motto: "*let the symbols do the work*". This feature allows working with uninterpreted formulae and manipulate them according to their symbols and associated rules. Also, it shows the dynamics of the symbols, avoids the introduction of syntactic errors, and requires less effort from the user.

As an example, consider the boolean expression $a{\vee}(b{\wedge}c)$. As $\vee$ distributes over $\wedge$, it can be useful to apply the distributivity rule automatically to obtain $(a \vee b) \wedge (a \vee c)$. Our library supports this type of manipulation.

Following the principles of the calculational method, our library has the general knowledge of how to apply rules but does not know if applying a certain rule is semantically correct. Checking the validity of applying a rule is the user's responsibility. We want the user to be in control and not the tool.

The main actions supported by the library are distributivity and factorization, associativity, symmetry, group and ungroup[5], and substitution of equals for equals. This ensures some of the reliability and flexibility that we want to provide. The rules supported consist purely of syntactic manipulation since the aim of the library is to support the calculational method.

One novel feature of the library is that everything on the screen remains handwritten, i.e., there is no conversion of the handwriting to typeset. Even the results of manipulations are handwritten. Tools like *MathPaper* [25] have the option of leaving the original input handwritten; however, the result of manipulating that input is presented in typeset. Our editor differs from that by keeping the original input and all the results of manipulations in their handwritten form. To show handwritten results, manipulation rules use the ink of the formulae being manipulated and rearrange it as needed. If a manipulation action needs a character that is not present in the formulae being manipulated, it will retrieve its handwritten representation from a database of characters defined by the user.

For most manipulation rules, gestures are used to trigger them. In the next section, we give more details on this.

Use of gestures

Gestures are a straightforward way of triggering actions. Using a gesture instead of a menu option is quicker and less interruptive. One of our main goals is to provide mechanisms that interfere as little as possible with the user's thought. For that reason, gestures are used to perform editing tasks and to manipulate algebraic formulae. For each action we provide default gestures that, in our view, are intuitive for that particular task. We also provide a gesture editor that allows the user to change the association between gestures and actions.

Examples of gestures that we have defined for editing tasks are the *circle* gesture, which is used to select the content inside the circle, and *right-down* and *right-up* gestures, which are used to add and remove vertical space (see Figure 6).

[5]The actions of grouping and ungrouping can be used to change the precedence of operators. For example, if given $a{\times}b + c{\times}d$ we ungroup the sub-expression $a{\times}b$, the first $\times$ and the $+$ will have the same precedence. To return to the original expression, we can group the sub-expression $a{\times}b$.

(a) Circle. (b) Right-down. (c) Right-up.

Figure 6. Gestures for editing tasks.

To apply mathematical rules to expressions, gestures are also used. Examples of two gestures that we use to apply the distributivity rule are depicted in Figures 7.a and 7.b. In this case, the distributivity of boolean disjunction through conjunction is used ($a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$). In Figure 7.a, the *semi-circle* gesture (drawn from left to right) is used to indicate that the disjunction is to be distributed over conjunction. The result presented in Figure 8 is then automatically obtained. Similarly, in Figure 7.b, the symbol $\vee$ is selected, dragged, and then dropped over the $\wedge$ symbol so that the first distributes over the second. Again, the result (Figure 8) of applying the rule is automatically obtained.

(a) Semi-circle (b) Drag-and-drop

Figure 7. Gestures for mathematical rules.

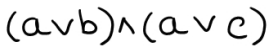


Figure 8. Result of applying the distributivity rule.

Some other commonly used rules, like symmetry, have gestures associated with them. We believe that the use of gestures is a good way to illustrate the dynamics of the symbols and to help the user writing calculations.

Animations of certain algebraic rules

Applying algebraic rules automatically and obtaining immediately a final result is helpful while solving mathematical problems. However, in a teaching environment, showing the final result of applying a rule may not be enough. When one sees a rule in action for the first time, one might need to see how the rule works in more detail in order to understand what it does. For that reason, we provide an option of animating the application of rules. Animations consist in moving slowly the several elements that form the formulae until they reach their position in the final result. As suggested in [6] and [15], animation and visualization techniques can improve student learning. This confirms our belief that animations can help the students keep track of the steps involved in the calculations.

Automatic layout adjustment

Often, while writing a document, the user has to adapt the layout of what is already written to allow more space to keep writing. This type of action interrupts the user's thought. Our library supports automatic space adjustment when the user needs it (the current version only supports quantifiers written in the Eindhoven notation [10]). For example, if the user is editing the summation quantifier shown in Figure 9, the characters marked in red are adjusted automatically as the user writes inside the quantifier expression.

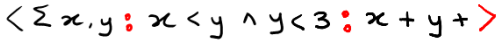
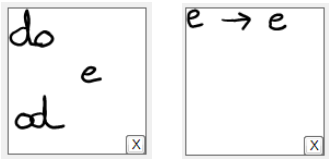


Figure 9. Summation quantifier.

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. In the example of Figure 9, the expression $x{+}y{+}$ is unfinished, so if one starts to write the rest of the expression, the '$\rangle$' symbol will move to the right, allowing space to finish it.

Handwritten templates

The MST library contains a mechanism for handwritten templates. Users are able to store expressions with structure and use them whenever they want. These can be seen as templates and, using substitution of equals for equals, can be composed to obtain more complex expressions. For example, the user may define templates for the GCL language [10] that can be reused later, avoiding the burden of having to recognize it again. Figure 10 shows two handwritten templates. The expressions denoted by *e* in the figure are placeholders that can be replaced by other expressions. The idea is that templates can be composed to write complex expressions/algorithms.



(a) do .. od (b) Guard

Figure 10. Two handwritten templates for the GCL language.

INTEGRATION WITH OTHER TOOLS

The MST library was designed to be easily added to other tools. For example, we have extended Classroom Presenter [1] (CP), an award-winning Tablet PC tool for ink-based teaching presentations and classroom interaction. Although we had no documentation about *CP*, we were able to use our library and its features within this tool.We only had to import the library and extend the user interface to allow the use of the imported features. Thus, we believe that, given appropriate documentation, other programmers can easily extend existing tools to make use of our library's features.

Using the Extended Classroom Presenter

In this section, the proof of theorem (1) shown previously is presented to demonstrate how the features available can be used to do mathematics. The gestures defined by default are used. Recall that the theorem to prove is:

$$(m{\times}p)\nabla n = m\nabla n \quad \Leftarrow \quad p\nabla n = 1 \quad .$$

The full proof 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 user starts by writing the theorem and recognizing it. The next step is to copy $m \nabla n$. For that, the user selects it (double-clicking on the second ∇):

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

and copies it using the gesture *Check*:

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

The result is:

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

In this example, the *Check* gesture is overloaded. It is used for copy and for substitution of equals for equals (also called *Leibniz* rule). Next, the hint for the step is written. For that, the user can select the expression $p \nabla n = 1$:

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

and use the *Check* gesture, once again, to copy:

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

The user can extend the hint with handwritten text. Next, the initial expression $m \nabla n$ is selected:

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

and copied:

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

The new copy is manipulated as shown in Figure 2. The user starts by replacing m by $m \times p$. So, $m \times p$ is selected:

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

and *Leibniz* is applied to m:

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

The result is:

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

After a number of steps to replace p by $p \nabla n$ and add brackets (omitted), we reach a point where distributivity is applied (using the semi-circle gesture):

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

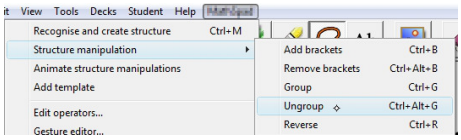
The result is the following:

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

The editor allows the user to remove the brackets surrounding the expression $(m \times p) \nabla (m \times n)$ (steps omitted). After removing the two sets of brackets, it is not possible to select the sub-expression $(m \times n) \nabla n$, because the expression $(m \times p) \nabla (m \times n)$ remains grouped. To proceed with the calculation, the user has to ungroup it. That is done by selecting the expression:

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

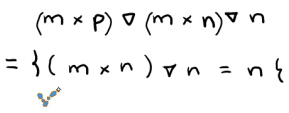
and using the ungroup option from the menu:



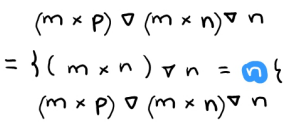
Now, the following step consists of replacing $(m \times n) \nabla n$ by n. Omitting the steps used to write the hint, the next step is to select the expression that was obtained in the previous step of the calculation:

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

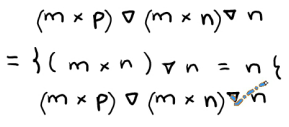
and then, copy it:



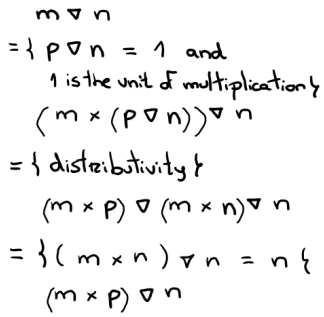
Finally, the hint is used to replace $(m \times n) \nabla n$ by n. First, select n:



and then using *Leibniz*:



The final result is obtained and the proof is concluded. The complete proof is as follows:



We believe that this proof illustrates well the advantages of having a structure editor of handwritten mathematics for presentations. Not only the steps are made easier, but also the user can be sure that manipulation errors will be avoided, since the tool will perform the manipulations for them. Another advantage that is relevant in a teaching context is that the use of rules is explicit. For example, to select the subexpression $(m \times n) \nabla n$, the user is forced to ungroup the expression $(m \times p) \nabla (m \times n)$; this corresponds to the use of associativity, which is often missed by students.

EVALUATION AND USER FEEDBACK

To gain feedback about the concept and functionality of the MST editor, we invited computer science teachers/researchers and students to trial the software. One of the main goals was to check if the features provided were regarded as helpful in the context of teaching using the calculational style. The extended version of *CP* was used for the evaluation. The tests were designed so that teachers would experiment with the software as a presenting tool and the students would experience it as a learning tool. Students did not use the software. 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. Although the editor can be used by students to assist them in the learning process, it was created with the teaching process as a main concern, i.e. with teachers as its main target *users* and students as *observers*.

In the teachers group there were 8 (eight) testers. These were teachers/researchers and PhD students who have previously been exposed to the calculational method. This group was given a guide detailing some steps that they had to follow in order to write a proof using the system (the one shown in the previous section). A questionnaire was given to record their opinion and they were observed by a monitor.

In the students group there were 74 (seventy four) second-year undergraduate students. All students had already been exposed to the calculational method. The test consisted of a 20-minute presentation followed by an anonymous questionnaire. The presentation started with an introduction to the system and an overview of its features. The system was then used to present the calculational proof shown in the previous section as it would be done in a lecture.

The results obtained in both tests 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.

Of particular interest is the task given to the teachers, where, for each step, they were 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[6]. However, they soon got used to the software/pen and performed the remainder of the test without problems.

In Table 1 the average results reported for the features used during the task are presented[7]. 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).

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

Table 1. Level difficulty reported (0 - Very easy; 4 - Very difficult)

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

[6]Most problems were due to the users being unfamiliar with the use of Tablet PCs.

[7]The average results are rounded to two decimal places.

check over a symbol; the first point of the hand-drawn check symbol 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.

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.

The results also show that teachers find the software useful for teaching and students find it useful for learning. For instance, teachers were asked to give their opinion about the statement "The features provided by this software can help with teaching mathematics". Clearly they found it useful as 5 (62.5%) of them do agree and 3 (37.5%) do strongly agree. One of the students' tasks was to comment on the statement "Did the software help you understanding the manipulations involved?"; 63 (85%) answered Yes whilst only 8 (11%) answered No; 3 (4%) students did not answer. This shows that, for this proof, the software has definitely helped them understanding the steps involved. Thus, the editor seems to be regarded as a helpful contribution to teaching mathematics in a calculational style. The use of gestures to trigger actions was well received by both teachers and students. When asked to comment the statement "Gestures are a good way to trigger actions", only one student disagreed, three students were neutral, and the remainder of the students and all the teachers agreed. We believe this result is due to the kinesthetic nature of gestures, which helps students visualize syntactic manipulations. When students were asked if they would use the software for other purposes, 38 (51%) answered Yes, 28 (38%) answered No, and 8 (11%) did not provide an answer. Some of their suggestions for other uses include verification of proofs, mathematics in general, and physics.

Both groups were asked for suggestions for improving the software. The suggestions included displaying the syntax trees of expressions, linking gestures to a sequence of actions, manipulating brackets by using gestures, connecting with theorem provers, adding replay feature for sessions, suggesting applicable rules, and further developing the system so that students can receive support from the teacher when away (including chat/video calls).

DISCUSSION AND CONCLUSION

The use of Tablet PCs for teaching and presentations brings many advantages. We currently use a Tablet PC to teach algorithmic problem solving to first-year undergraduate students and, according to their feedback, they enjoy the interactivity that comes with the use of the Tablet PC. Based on the evaluation of the system, we believe that our library will further improve this interactivity.

The benefits of presenting the mathematics involved in algorithmic problem solving with the features provided can be enormous. It is possible to demonstrate the dynamics of problem-solving in a manner that improves on blackboard-style of teaching by exploiting the reliability of computer software in copying and manipulating structured information. It is possible to show in real-time the steps of a calculation and to see how the expressions are transformed from one step to the other.

From our experience, using a Tablet PC only as a device to write on does not exploit its full potential. The Tablet PC's computational capabilities can be a valuable help to write calculations and to show the dynamics of the symbols (that the calculational method stresses so much). Even if the presenter does not want to emphasize syntactic manipulations, having, for example, a feature that allows reliable copy of expressions is a huge improvement.

All the features presented in this paper contribute, in our view, to the flexibility and reliability that we want to provide and, more importantly, they help in doing mathematics without interfering with the user's thought. Although this library was created with presentations in mind, it can also support students and researchers in solving mathematical problems.

The evaluation of the library with teachers and students has confirmed that the features available are usable, suitable for their purpose, and useful for teaching and learning. However, it has also shown that familiarity with the use of Tablet PCs and with the features of the tool are important usability factors. In the future we expect to carry out an experiment where students use the tool to handwrite and manipulate mathematical content (we also plan to use large surfaces, such as walls and tables).

It is important to note that our library is just a prototype. There are still many challenges ahead, the most important being the improvement of recognition of handwritten mathematics. Even though recognition of handwritten mathematics is outside the scope of this work, having a recognizer oriented to the mathematical structures used in teaching algorithmic problem solving in a calculational style would make the editor more usable. Moreover, the current functionalities can still benefit from more improvements and testing. After a more stable library is achieved, our next step will be to look into connecting it with a theorem prover so that the system can produce automated proofs and support the user in writing a proof by suggesting valid steps. Another avenue that we may pursue is the support for algorithm animation. It would also be interesting to investigate how the features described in this paper could be integrated into systems that support more advanced interactions (like Hands-On Math [26]).

Even though this is not a professional tool, it can still be used for presentations and it is a starting point for what can become a very useful tool.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for the helpful comments. The work presented in this paper was supported by FCT (Portuguese Foundation for Science and Technology) with grant SFRH/BD/29553/2006.

REFERENCES

1. Anderson, R., Anderson, R., Chung, O., Davis, K. M., Davis, P., Prince, C., Razmov, V., and Simon, B.

Classroom presenter - a classroom interaction system for active and collaborative learning. In *WIPTE* (2006).

2. Backhouse, R. *Program Construction*. John Wiley & Sons, 2003.
3. Backhouse, R. *Algorithmic Problem Solving*. John Wiley & Sons, 2011.
4. Backhouse, R., and Verhoeven, R. Mathspad: A system for on-line preparation of mathematical documents. *Software - Concepts and Tools 18* (1997), 80–89.
5. Bott, J. N., and LaViola Jr., J. J. A pen-based tool for visualizing vector mathematics. In *SBM*, M. Alexa and E. Y.-L. Do, Eds., Eurographics Association (2010), 103–110.
6. Catrambone, R., and Seay, A. F. Using animation to help students learn computer algorithms. *Human Factors: The Journal of the Human Factors and Ergonomics Society 44*, 3 (2002), 495–511.
7. Cossairt, T. J., and LaViola Jr., J. J. Setpad: A sketch-based tool for exploring discrete math set problems. In *SBM*, K. Singh and L. B. Kara, Eds., Eurographics Association (2012), 47–56.
8. Cromack, J. 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).
9. Dijkstra, E. W. Indirect equality enriched (and a proof by Netty). Available from `http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1315.PDF` [Last accessed: 29 June 2014], December 2001.
10. Dijkstra, E. W., and Scholten, C. S. *Predicate Calculus and Program Semantics*. Springer Verlag, NY, 1990.
11. Ferreira, J. F., and Mendes, A. Student's feedback on teaching mathematics through the calculational method. In *39th ASEE/IEEE Frontiers in Education Conference*, IEEE (2009).
12. Fitzgerald, T. J. The Tablet PC takes its place in the classroom. The New York Times, 9 September 2004. Available from `http://www.nytimes.com/2004/09/09/technology/circuits/09jott.html` [Last accessed: 29 June 2014].
13. Graham, R. L., Knuth, D. E., and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1994.
14. Gries, D., and Schneider, F. *A Logical Approach to Discrete Mathematics*. Springer Verlag, NY, 1993.
15. Grissom, S., McNally, M. F., and Naps, T. Algorithm visualization in CS education: Comparing levels of student engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, ACM (New York, NY, USA, 2003), 87–94.
16. Labahn, G., Lank, E., MacLean, S., Marzouk, M. S., and Tausky, D. Mathbrush: A system for doing math on pen-based devices. In *Document Analysis Systems*, K. Kise and H. Sako, Eds., IEEE Computer Society (2008), 599–606.
17. LaViola Jr., J. J., and Zeleznik, R. C. Mathpad^2: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph. 23*, 3 (2004), 432–440.
18. Li, C., Miller, T. S., Zeleznik, R. C., and LaViola Jr., J. J. Algosketch: Algorithm sketching and interactive computation. In *SBM*, C. Alvarado and M.-P. Cani, Eds., Eurographics Association (2008), 175–182.
19. Simon, B., Anderson, R., Hoyer, C., and Su, J. Preliminary experiences with a Tablet PC based system to support active learning in computer science courses. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, ACM (New York, NY, USA, 2004), 213–217.
20. Smithies, S., Novins, K., and Arvo, J. Equation entry and editing via handwriting and gesture recognition. *Behaviour & IT 20*, 1 (2001), 53–67.
21. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., and Kanahori, T. INFTY: an integrated OCR system for mathematical documents. In *ACM Symposium on Document Engineering*, ACM (2003), 95–104.
22. Verhoeven, R., and Backhouse, R. Towards tool support for program verification and construction. In *FM'99 - Int. Formal Methods Symposium*, J. W. Jeanette Wing and J. Davies, Eds., Springer Lect. Notes Comp. Sci. (1709) (1999), 1128–1146.
23. Wang, M., Plimmer, B., Schmieder, P., Stapleton, G., Rodgers, P., and Delaney, A. SketchSet: Creating Euler diagrams using pen or mouse. In *VL/HCC*, G. Costagliola, A. J. Ko, A. Cypher, J. Nichols, C. Scaffidi, C. Kelleher, and B. A. Myers, Eds., IEEE (2011), 75–82.
24. Wenzel, L., and Dillner, H. MathJournal - an interactive tool for the Tablet PC. Available from `http://www.xthink.com/downloads/MathJournal_2003.pdf` [Last accessed: 29 June 2014].
25. Zeleznik, R., Miller, T., Li, C., and Laviola, Jr., J. J. Mathpaper: Mathematical sketching with fluid support for interactive computation. In *SG '08: Proceedings of the 9th international symposium on Smart Graphics*, Springer-Verlag (Berlin, Heidelberg, 2008), 20–32.
26. Zeleznik, R. C., Bragdon, A., Adeputra, F., and Ko, H.-S. Hands-On Math: a page-based multi-touch and pen desktop for technical work and problem solving. In *UIST*, K. Perlin, M. Czerwinski, and R. Miller, Eds., ACM (2010), 17–26.