

# Proposal for Coexistence of Mathematical Handwritten and Keyboard Input in a WYSIWYG Expression Editor

Juan Lao-Tebar  
juan@wiris.com

Francisco Álvaro  
falvaro@wiris.com

Daniel Marquès  
dani@wiris.com

WIRIS math, Barcelona, Spain

## Abstract

Despite math notation is well-known and extensively used in many fields, introducing math expressions into a computer device is not straightforward and requires specific notations. For this reason, users commonly use editors that make this task easier, and even handwriting can be used directly as input method. In this paper, we propose a design of interface and behaviour for the coexistence of a keyboard-based mathematical editor and a handwritten mathematical expression recognizer. Advantages of both modes are analyzed and it is shown that they are complementary, so a design based on coexistence is more appropriate for the final user than single modes. Also, regarding accessibility support, we provide some hints and guidelines on an implementation that offers a good experience to users with disabilities.

## 1 Introduction

In recent years, the World Wide Web Consortium (W3C) and major web browsers have invested heavily in developing and expanding web standards to provide a greater separation between content and presentation to improve user experience. Specifically, HTML5 introduced new types of input fields for web forms. In the new standard, an input field can be used to enter dates, time, colors, emails, phone numbers, ranges or even something as simple as a floating number, taking in account users local designation of decimal mark.<sup>1</sup>

Years ago, if a developer needed to use one of those input types in a web form, the developer had to include javascript code in the page. (S)he had to write his own code or use third-party libraries. This caused several problems to the user. Since every page had to implement their own way of entering special input types, cohesion in the user interface and behaviour did not exist. In addition, language support, accessibility features and responsiveness depended on each implementation.

Web browsers do not have a native mathematical input field for web forms, since HTML5 specification does not define it.<sup>2</sup> In addition, due to the complexity of the edition of mathematical expressions (and even if HTML5 defined a mathematical input field), the existence of third-party solutions would improve the user experience and notation coverage. Something similar to the case of rich-text editors, where third-party solutions such as TinyMCE<sup>3</sup> or CKEditor<sup>4</sup> are used to provide a full featured web rich-text editor since W3C and web browsers

---

*Copyright © by the paper's authors. Copying permitted for private and academic purposes.*

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at <http://ceur-ws.org>

<sup>1</sup><https://www.w3.org/TR/html5/>

<sup>2</sup><https://www.w3.org/TR/html-markup/input.html>

<sup>3</sup><https://www.tinymce.com/>

<sup>4</sup><http://ckeditor.com/>

do not provide a standardized solution for this requirement.

There are different libraries available to deal with math expressions in previously discussed scenarios. Some of them let the user insert expressions using L<sup>A</sup>T<sub>E</sub>X, like WikiEditor<sup>5</sup> (content editor for Wikipedia), while other libraries offer a more user-friendly interface, where what you see is what you get (WYSIWYG) and no previous knowledge of any language or format is needed. Examples of this kind of libraries are DragMath,<sup>6</sup> MathQuill,<sup>7</sup> CodeCogs<sup>8</sup> or WIRIS editor.<sup>9</sup>

With the rise of touch devices and artificial intelligence systems, new systems are appearing for recognizing handwritten mathematical expressions, like MyScript<sup>10</sup>, MathBrush [ML15], *m<sub>in</sub>* [SHP<sup>+</sup>12] or WIRIS hand.<sup>11</sup> These systems have some advantages compared to keyboard input: they offer a more comfortable interface on touch devices and they are more user-friendly and intuitive than classic overloaded toolbars that keyboard-based editors usually have. However, sometimes a user may prefer to insert expressions using the classic keyboard for several reasons:

- Lack of a touch device or not achieving the required accuracy with the mouse when drawing an expression.
- Suffering from visual impairment.
- In general, keyboard input offers a wider variety of mathematical expressions than a handwriting recognizer.
- Creation of expressions with a specific style and formatting (colors, position of elements, etc.).

For these reasons, the user should decide what input method (s)he wants to use, and should be able even to change it during the editing session.

In summary, as a developer, there should be a mathematical input field, independent to the input method (keyboard or handwritten); and as a user, the interface should offer a way to manage the input methods. Some vendors already implemented this duality in their systems, like Learnosity<sup>12</sup> and WIRIS.<sup>13</sup>

## 2 Keyboard Input Interface

A keyboard-based mathematical editor is traditionally composed by a toolbar and an editing area. Of course, this structure can vary between implementations by adding or removing interface elements but these variations are out of the scope of this paper. An example of this structure is shown in Figure 1.

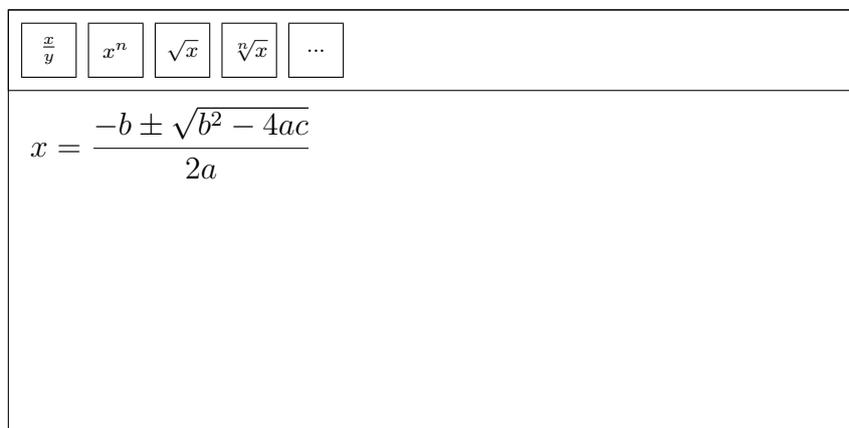


Figure 1: Basic structure of a keyboard-based mathematical editor

<sup>5</sup><https://www.mediawiki.org/wiki/Extension:WikiEditor>

<sup>6</sup><http://dragmath.bham.ac.uk/demo.html>

<sup>7</sup><http://mathquill.com/>

<sup>8</sup><http://www.codecogs.com/latex/about.php>

<sup>9</sup><http://www.wiris.com/editor/demo/en/>

<sup>10</sup><http://webdemo.myscript.com/#/demo/equation>

<sup>11</sup><http://www.wiris.com/hand>

<sup>12</sup><https://www.learnosity.com/>

<sup>13</sup><http://www.wiris.com/editor/demo/en/>

The toolbar of a keyboard-based mathematical editor is used for changing content formatting, switching element properties, managing clipboard (copy, cut, paste), performing undo/redo actions and inserting mathematical structures that cannot be typed using a keyboard (like fractions or square roots), among other features.

A keyboard-based mathematical editor also offers advanced features that are difficult to implement in a handwriting recognition system, such as support for different character sets (chinese, japanese, korean, arabic), Right To Left (RTL) support, syntax checking, copy/cut/paste, formatting or pixel-perfect element position. In summary, a keyboard-based mathematical editor is a powerful tool that can be used not just to let the user insert expressions, but also to decide exactly how they should look.

### 3 Handwritten Input Interface

It is difficult to determine the interface structure of a handwritten mathematical expression recognizer, since the technology is still new and there are different implementations in the market without any established design pattern.

The common characteristic in most of them is that there are two main components: the drawing area and the action buttons, that can be displayed inside a tiny toolbar (Figure 2) or floating over the drawing area (Figure 3).

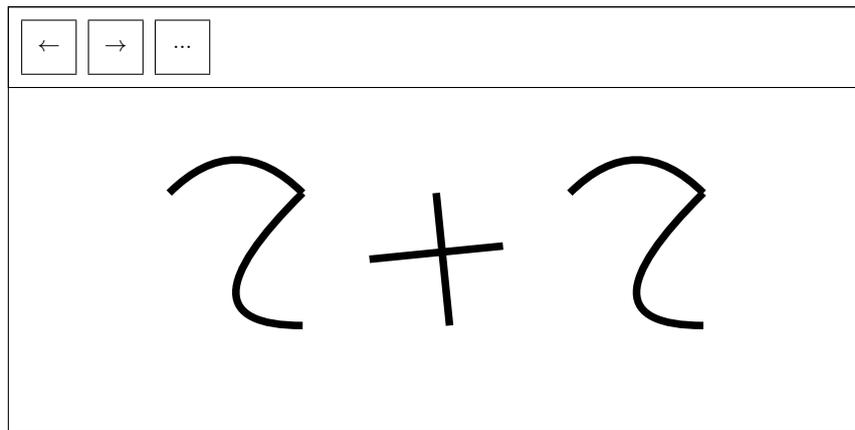


Figure 2: Basic structure of a handwritten mathematical expression recognizer with toolbar

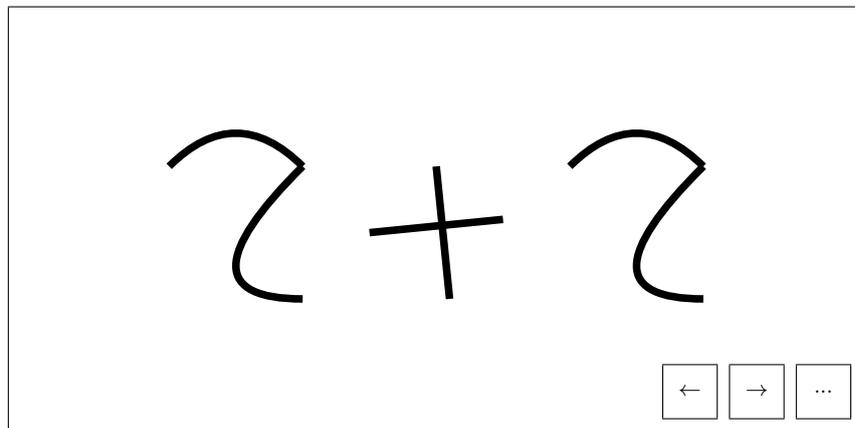


Figure 3: Basic structure of a handwritten mathematical expression recognizer with floating buttons

Action buttons can be used to perform simple tasks, as undo/redo actions or clearing the drawing area.

The drawing area allows the user to write mathematical expressions using a touch screen or a mouse, but it can be used also to enter gestures. A gesture is a special input that the system interprets as a command. For example, drawing several strokes over an existing drawn character can be interpreted as a desire to delete the

character (Figure 4); or touching with two fingers at the same time and changing the distance between them can be interpreted as a desire to increase/decrease the zoom of the drawing area.

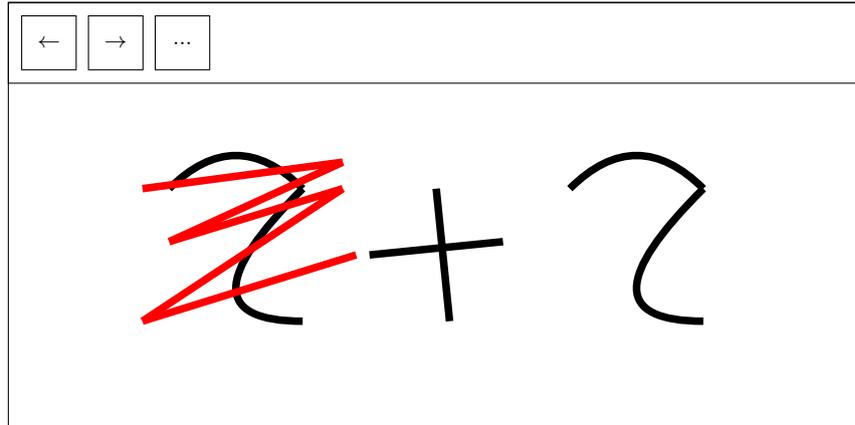


Figure 4: Deletion gesture

This kind of interface is simpler than the interface of a keyboard-based mathematical editor. It can be useful in environments where the semantic of the entered value is more important than the formatting.

## 4 Coexistence of Handwritten and Keyboard Input

As said previously, both interfaces have different and complementary characteristics. In some scenarios, just one of them is enough to fulfill the given specific requirements, but in other cases the final decision is up to the user.

In this paper, we propose the coexistence between both interfaces. On the startup, the system detects and uses the appropriate interface depending on the device, user preferences or developers decision, but the user can interchange the interface at any moment, even during the editing session.

### 4.1 Interchange of Interface

There are several ways to interchange between a keyboard-based mathematical editor interface and a handwritten mathematical expression recognizer interface. In this paper, we suggest the usage of an element that indicates clearly the concept of duality, avoiding approaches based on gestures like swiping without any previous visual clue. The described proposal can be implemented via toolbar (Figure 5) or float buttons.

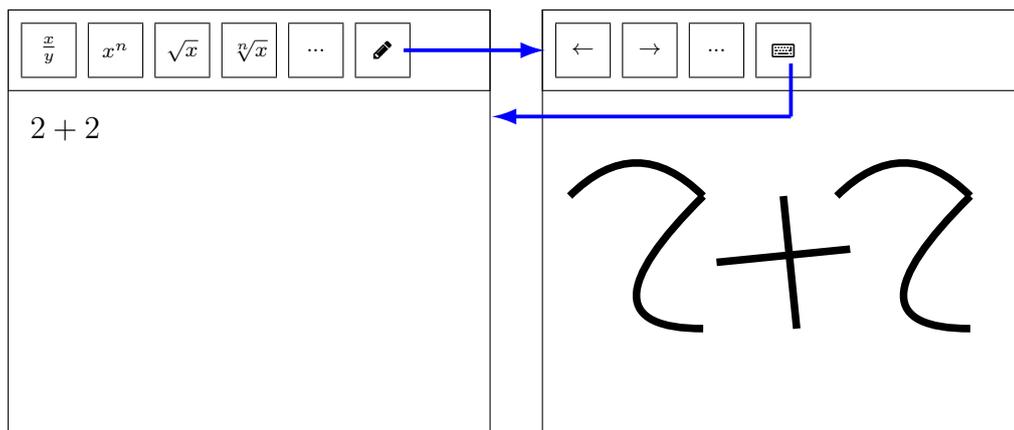


Figure 5: Interchange between a keyboard input interface and a handwritten input interface via toolbar buttons

In this proposal, we also recommend the conservation of the existing content (if there is any) as long as possible, converting it from one input method to another, as explained below.

## 4.2 Conversion of Formal Content to Strokes and Vice Versa

Given a handwritten math input, once recognized, its conversion to a formal content for the keyboard-based mathematical editor is trivial: the formal result of the recognizer is just passed to the editor, assuming that all the pieces use the same format.

The conversion of content typed with a keyboard-based editor to a set of strokes understandable by the handwritten expression recognizer is not immediate, because it is necessary to create a correct 2D structure containing handwritten symbols. The technical details are out of the subject of this paper, but one possible solution can consist on reusing the existing painting technology of the WYSIWYG editor, sending the paint instructions directly to the drawing area of the recognizer.

For the conversion of text characters to strokes, an existent font made with drawing instructions can be used, and the system just have to execute the corresponding paint instructions when a character must be drawn. A real implementation of this proposal has been done in the WIRIS editor. Figure 6 shows the automatically generated handwritten expression.

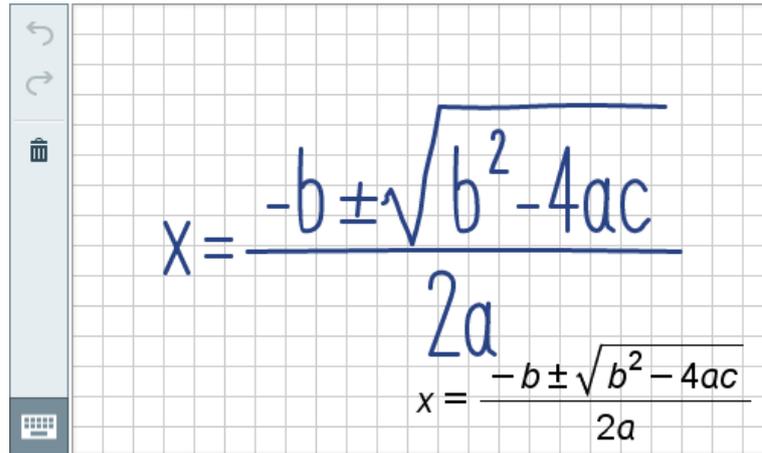
The image shows a screenshot of the WIRIS editor interface. On the left, there is a vertical toolbar with icons for undo, redo, delete, and a keyboard icon. The main area is a grid where the quadratic formula  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$  is written in blue ink. Below the main formula, the same formula is shown in a smaller, more formal font, representing the formal content generated from the handwritten input.

Figure 6: Automatically generated handwritten expression in WIRIS editor

## 4.3 Developer Tools

In this section, we present some guidelines for API development where keyboard and handwriting inputs coexist.

### 4.3.1 Providing Clues and Hints About the Expected Value

In some platforms, like in assessment environments, the mathematical input field is used to let students provide an answer to a problem or question.

Since recognition of handwritten math expressions is not 100% accurate and due to handwriting ambiguities, it is recommended to pass hints and clues to the recognizer in order to increase the success rate.

For example, if the platform knows that the student has to give an answer that is composed just by numbers, notifying it to the recognizer can increase the success rate since the recognizer will not classify confusing drawings of “2” like “z”.

Technical details about the format of those clues and hints go beyond the purpose of this paper.

### 4.3.2 Obtaining Information About the Input Mode

In some cases it is interesting for the platform to store the drawing of the user. For example, in assessment environments, when the student answers using the handwriting recognizer and the system is unable to recognize the expression with success, a teacher can inspect manually the strokes drawn by the student and determine if the answer is correct or not.

In some formats like MathML it is possible to insert semantic content next to the formal representation of the expression (in this case, using a semantics tag<sup>14</sup>). In other formats like  $\LaTeX$ , this semantic content can be

<sup>14</sup><https://www.w3.org/TR/REC-MathML/>

inserted just as a  $\LaTeX$  comment, and the real code that represents the expression is not affected by it.

This paper proposes the insertion of the user drawing as a semantic content inside the returned value when the developer explicitly retrieves it in this way from the input field.

In the future, if new input methods are implemented (like voice recognition), other semantic content can be returned next to the real representation of the expression using this method.

### 4.3.3 Interface Events

An API for managing interface events increases extensibility and adaptation to customer requisites, and allows the developer to have full control on the input field behaviour. This paper proposes the following events, not entering in details about the technical implementation. In general:

- An event that is fired when the value of the field changes.
- An event that is fired when the interface input mode changes.

For the keyboard-based mathematical editor:

- An event that is fired when the selection or caret position changes.
- An event that is fired when the formatting of the expression or the selection changes.

For the handwritten mathematical expression recognizer:

- An event that is fired when the drawing area changes.
- An event that is fired when the system recognizes an expression.
- An event that is fired when the system cannot recognize an expression.

With these events a developer can, for example, prevent the submission of a form if the entered strokes have not been still recognized (or there has been an error in the recognition), in order to prevent wrong submissions.

## 4.4 Default Interface Mode on Startup

On the startup the system has to decide the appropriate interface to be displayed for entering mathematical expressions. This decision is based on several factors.

The developer has all the power to decide what interface must be used, since the developer knows the context of the page and, for example, if the entered value is going to be evaluated later or is going to be inserted in a page with a specific formatting. The developer knows also the main user target (primary education, secondary education, university students, etc.), and can decide that a complex expression should be entered just with the keyboard-based editor (like, for example, a Taylor series development).

If the developer does not take a decision over the default interface, there are several scenarios. If there are stored user preferences from previous sessions, the system can load them and display the default interface based on the current configuration.

If a user configuration is not defined, this proposal suggests that the system should determine, as far as possible, if the platform expects a given value and the user can use the handwriting recognizer to enter it.

In other words: if the handwriting recognizer is not able to recognize the expected value, the system must display the keyboard-based editor interface by default (and in some cases, it is recommended to disable completely the handwritten input). Of course, in this scenario the developer needs to provide a clue about the expected value, as seen previously.

Otherwise, the system can detect characteristics from the device. For example, if the user is using a device with a touch screen or a digital pen, the handwriting recognizer can be used as default interface. The system can detect other properties, like if the user is using a screen reader due to a visual impairment and, in that case, enable the keyboard-based editor by default.

## 5 Accessibility

Web accessibility refers to the inclusive practice of removing barriers that prevent interaction with, or access to websites, by people with disabilities. When sites are correctly designed, developed and edited, all users have equal access to information and functionality.

The needs that Web accessibility aims to address include:

**Visual** Visual impairments including blindness, various common types of low vision and poor eyesight, various types of color blindness.

**Motor/mobility** e.g. difficulty or inability to use the hands, including tremors, muscle slowness, loss of fine muscle control, etc., due to conditions such as Parkinson’s Disease, muscular dystrophy, cerebral palsy, stroke.

**Auditory** Deafness or hearing impairments, including individuals who are hard of hearing.

**Seizures** Photo epileptic seizures caused by visual strobe or flashing effects.

**Cognitive/Intellectual** Developmental disabilities, learning disabilities (dyslexia, dyscalculia, etc.), and cognitive disabilities of various origins, affecting memory, attention, developmental “maturity”, problem-solving and logic skills, etc.

By the nature of a mathematical input field and the subject of this paper, only visual and motor/mobility disabilities are taken in account in this proposal.

In the case of motor/mobility disabilities, the major part of the work is already performed by the web browser, the operating system and third-party native tools, performing the required zoom on the page and enabling special features on mouse and keyboard (like enabling special keys or modifying the behaviour of classical mouse drag&drop). In this paper we assume that these features are enough to offer a good experience to motor/mobility disabled users and It does not propose any extra work. But, of course, implementations are free to include new features that help on the removal of this barrier.

In the case of visual disabilities, due to the nature of a handwriting recognizer, this proposal recommends the exclusive usage of a keyboard-based mathematical editor, which can include more advanced features for visual impaired users than a handwriting recognizer, like keyboard navigation and support for screen readers.

### 5.1 Keyboard Navigation

On the keyboard-based editor, the system must implement a way of accessing all elements (toolbar, buttons, editing area and others) using just the keyboard. This feature can be already implemented using the focus property defined by W3C.<sup>15</sup>

When a user activates the button that changes the interface to the handwriting recognizer, we propose to make focusable just the button to go back to the keyboard-based editor. By this, a user that suffers from visual impairment or a motor/mobility disability can go easily back to the classical editor that offers full accessible support.

### 5.2 Screen Reader Support

Modern accessible web technologies offer advanced support for screen readers, allowing the developer decide what the screen reader should synthesize. On the keyboard-based mathematical editor, this feature can be used for two purposes:

- Providing description of interface elements when they are focused, like tabs, action buttons and editing area.
- Providing a description of the context when the caret is moved, like in a classic input field. When the caret is moved in a classic input field, the screen reader usually spells the name of the character at the caret position (or other details, like if the caret is at the end of a line). This feature can be also implemented in a keyboard-based mathematical editor using the standard aria-live region,<sup>16</sup> providing a brief description of the context of the caret (beginning of a square root, denominator of a fraction, etc).

---

<sup>15</sup><https://www.w3.org/TR/html5/editing.html#focus>

<sup>16</sup>[https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA\\_Live\\_Regions](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Live_Regions)

## 6 Examples and First Integrations

The model of interface proposed in this paper has been already implemented in WIRIS quizzes. WIRIS quizzes (Figure 7) is a software component that allows teachers to create questions for students, containing random variables, providing automatic evaluation of answers and using WIRIS editor as mathematical input field for students.

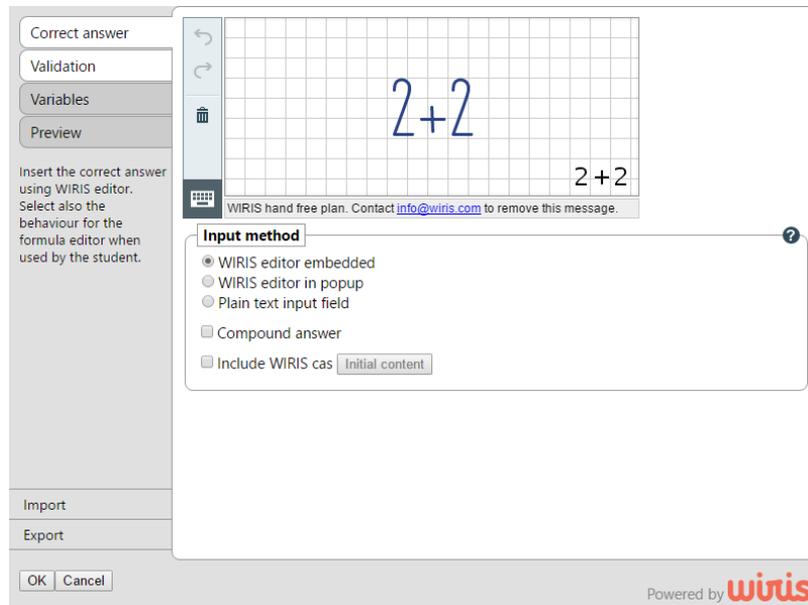


Figure 7: Question creation in WIRIS quizzes

When a teacher creates a question and provides the structure of a correct answer (e.g. specifying what variables, numbers or symbols are involved in it), WIRIS quizzes analyzes it and passes a set of constraints to WIRIS editor to increase the success rate of the handwriting recognizer. For example, if the correct answer contains only numbers and does not contain any “z”, the ambiguous strokes that might either be a number “2” or a letter “z” will be recognized as numbers. Figure 8 shows an example of failed recognition due to unknown information about the context, while Figure 9 shows how, with information, the recognizer has a higher success rate.

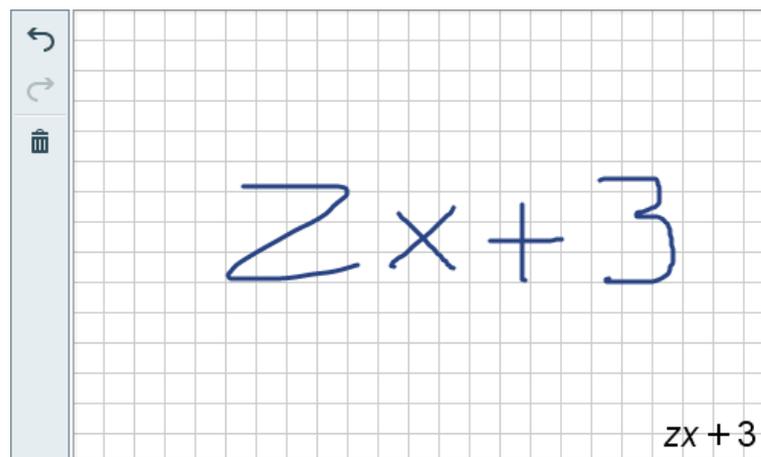


Figure 8: Failed recognition of number “2” without context information about the expected input

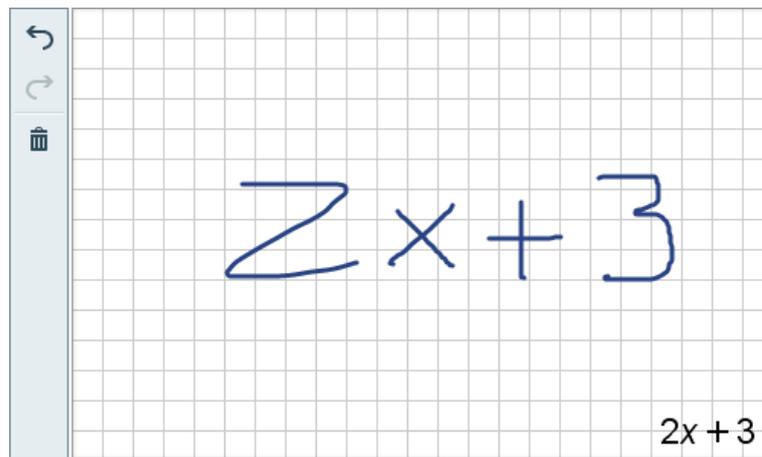


Figure 9: Once the system has information about the context, it recognizes correctly the character “2”

## 7 Discussion and Future Work

### 7.1 Technical Details and Standardization

This paper is a proposal for coexistence of mathematical handwritten and keyboard input in a WYSIWYG expression editor, and is not an API proposal or a definition of a standard.

From the point of view of the authors of this paper, the following subjects are still pending of discussion and standardization:

- Inclusion of a math type on standard HTML input field types.
- Definition of an API for the control of “math” input fields, including configuration parameters and event listeners.
- Election of a standard mathematical format for “math” input field values, like MathML.

### 7.2 Other Input Methods

Advancements in artificial intelligence and machine learning are promoting the development of new technologies that can analyze and classify information impossible to handle until now. Specially, advancements in ASR (Automatic Speech Recognition) introduce speech as a new input method of mathematical expressions [WHP<sup>+</sup>09]. There are even some studies that combine both handwriting and speech input to enter a mathematical expression [MMPVG13].

The interface proposal of this paper can be extended to these new input methods, adding a new button for interchanging the interface to the new input modes, extending the functionality of the existing interchanger button or adding a new button that just opens a modal dialog for speech input, like traditional text input already do (e.g. Google, Apple, Microsoft).

### 7.3 Drawing over the Current Formal Expression

Another possible approach for the problem handled in this paper can consist in allowing the user to draw new subexpressions over an existing formal expression in the mathematical editor. But this approach presents some problems:

- Using the mouse or the touch screen to handle the caret position and selection can conflict with the feature of drawing or performing gestures.
- Sometimes there is not enough space to draw a subexpression. For example, in this case it would be difficult to draw a new fraction inside the square root:

$$\frac{\sqrt{x}}{2}$$

- Recognizing incomplete subexpressions has lower success rate than recognizing complete expressions.

## 8 Conclusions

In this paper we proposed a design of interface and behaviour for the coexistence of a keyboard-based mathematical editor and a handwritten mathematical expression recognizer. Advantages of both modes have been analyzed and it has been shown that they are complementary, so a design based on coexistence is more appropriate for the final user than single modes.

In this proposal we also present some guidelines about the communication channel between developers and mathematical input fields that fulfills the needs a platform can have on a real scenario.

Regarding accessibility support, we proposed some hints and guidelines on an implementation that offers a good experience to users with disabilities.

Finally, future work should be focused on the definition of a standard API and format for real case scenarios and exposed.

## References

- [ML15] Scott MacLean and George Labahn. A bayesian model for recognizing handwritten mathematical expressions. *Pattern Recognition*, 48(8):2433–2445, 2015.
- [MMPVG13] Sofiane Medjkoune, Harold Mouchere, Simon Petitrenaud, and Christian Viard-Gaudin. Multimodal mathematical expressions recognition: Case of speech and handwriting. In *Human-computer interaction. interaction modalities and techniques*, pages 77–86. Springer Berlin Heidelberg, 2013.
- [SHP<sup>+</sup>12] Christopher Sasarak, Kevin Hart, Richard Pospesel, David Stalnaker, Lei Hu, Robert Livolsi, Siyu Zhu, and Richard Zanibbi. min: A multimodal web interface for math search. *Symp. Human-Computer Interaction and Information Retrieval, Cambridge, MA*, 2012.
- [WHP<sup>+</sup>09] Angela Wigmore, Gordon Hunter, Eckhard Pflugel, James Denholm-Price, and Vincent Binelli. Using automatic speech recognition to dictate mathematical expressions. *Journal of Computers in Mathematics and Science Teaching*, 28(2):177–189, 2009.