BrEdiMa: Yet Another Web-browser Tool for Editing Mathematical Expressions

Yasuhito NAKANO and Hirokazu MURAO

Department of Computer Science, The University of Electro-Communications

1 Introduction

In the last decade, communication via Internet has expanded explosively, and is still expanding. Most information transmitted and communicated is based on plain text, which can be input only with keyboard. On the other hand, mathematical expressions, which rarely appear in daily communication but are commonly used in education and research, are difficult to input, and consequently, they are not suited for daily and smooth communication. This situation motivated us to develop yet another tool to support input of mathematical expressions with editing facility. It should be of light-weight, easy to use or casually usable, and highly comformable to Internet and other Web tools. Some may think that TEX/LATEX is a common language to describe, and possibly, to communicate, mathematical expressions. However, input method commonly used by various word processing software is popular for most of users in the current network community. Input in WYSIWYG style will be preferred.

There are many tools developed for similar purposes and available in public, as listed later. Some of them are available as commercial products, and some of them require plugins for handling or display. Therefore, in order to introduce and use them, we usually have to go through some steps, which can often be a nuisance for novices. With this recognition in mind, we develop our own tool for Web browsers. Our tool is developed based on the following policy:

- free of charge,
- not requiring plugin, even on Microsoft Internet Explorer, and,
- editing with GUI in WYSIWYG style

JavaScript language [4] will be most appropriate for these purposes, and, above all, is easy to use. We stick to the use of JavaScript, keeping any extension with the new technology AJAX [5] in sight, as mentioned later. We shall use JavaScript in object-oriented style, retaining the close relation with MathML, including MathML output.

The existing tools take various forms depending on the methods for input and rendering. A key factor of the difference is the technology for rendering. It is closely related with which side of server or client to process, and also the language to use for implementation. A well-known technique used in mimeTeX [1] for rendering is the image generation by means of TEX via CGI. There are similar examples such as imgTeX [2] and texvc [3]. ASCIIMathML, implemented

in JavaScript, converts plain texts of mathematical expressions into MathML expressions and relies on browsers or plugins rendering. The W3C graphics standard SVG, supported by Firefox and a plugin by Adobe for Internet Explorer, is used by a GUI-editor sMArTH along with the application logic implemented in JavaScript. Flash can be an alternative technology for rendering and used by a commercial GUI editor MathIWYG. For more softwares and details, refer to the Web page of the W3C MathML software list[1] and the corresponding pages linked from the list.

2 Overall Design and Technologies Used

As a programming language, we use JavaScript. Programming in JavaScript enables us, without reloading Web-pages, to generate Web-pages dynamically on the client side, and to communicate with server, mainly for obtaining rendered images. Recently, triggered by the advent of GoogleMaps, a notion of AJAX (Asynchronous JavaScript and XML) has been newly introduced, and it revived the programming in JavaScript. If we develop our tool in JavaScript with AJAX-ready interface, we can use it as an user interface or a frontend with editing facility for other software, e.g., database retrieving system.

Although JavaScript is used in common to multiple kinds of browsers, its implementation is browser dependent due to slight differences. The target browser of our implementation includes Internet Explorer on Windows and Firefox.

While the whole document of a page is represented and handled dynamically as an object of DOM [6] by JavaScript, we represent mathematical formulas in it by objects of hierarchical structure, which just follows the structure of MathML's presentation markup. The structure defined by MathML provides sufficient capability to represent any mathematical expressions, and also makes output easy so that simply traversing the structure completes the output of mathematical expressions in our usual format.

The object-oriented property of JavaScript is termed "*prototype based*", and different from "*class based*" in that a new object is created from the existing one, not as an *instance* of a template *class*. As explained in the next section, we introduce a notion of class to represent a set of objects composed of multiple parts of mathematical expressions, by preparing an existent object used only for a template. Inheritance is implemented by preparing a function to copy some part of a parent object into the corresponding part of a child object.

With respect to rendering of mathematical expressions, we rely on the existing software; browsers' capability, maybe with the help of plugins, to render MathML, and the image generation by mimeTeX. Direct use of MathML is useful because it enables us the following.

(1) to change arbitrarily the size like other plain texts,
(2) to print in printers' full resolution, and
(3) to reuse and edit by dragging and copying by mouse.

[1] http://www.w3.org/Math/Software/

However, while graphic image is supported by every browser, the support of MathML is quite limited; Firefox, may requiring additional fonts, provides a good support, and Internet Explorer and Opera need the help of plugin, such as MathPlayer. We use both methods, and they are switched by users' choice. Correspondingly, we support output both in MathML format and in LaTeX. Rendering facility is also used for entity references of special symbols and extended characters which cannot be input direct from keyboard.

The final decision we made is on the kinds of mathematical expressions we handle. We treat only those elementary expressions which appear in textbooks of high-school level. Notations and symbols used in higher-level mathematics are not fixed and used freely (differ person-to-person). We may expect that those who require such special notations often be experts of TeX/LaTeX, and they are not the main target of our development.

3 Internals and Implementation

3.1 Objects

To represent mathematical expressions in JavaScript, we define four types of objects, MathInput object, MathRow object, Container object and Token object.

MathInput object corresponds to a token element of MathML and is a set of character objects which can be input direct from keyboard. Input is done via `textarea` markup as plain text. Key input is monitored to detect the characters `(`, `)`, `^` and `_` and convert them to the corresponding Container objects MathFenced, MathFenced, MathSup and Mathsub. Text strings corresponding to special symbols or extended characters, to be represented by MathString object, are also detected.

MathRow object corresponds to MathML's `mrow` element, and maybe to braces in LaTeX. Internally, every MathRow object holds an array of other kinds of objects, which compose a single mathematical expression.

Container object represents a class of objects that have mathematical structure such as rational expressions, and contains at least one MathRow object of a part of the structure. Every part is input as a corresponding MathRow object.

Token object represents such token elements of MathML that do not correspond to any single character input from keyboard and therefore, are hardly handled by MathInput object. In an HTML document, it generates a `SPAN` element, in which symbols or characters to display are contained. Token objects are further classified into MathSymbol and MathString objects. The former is for entity references which cannot be input direct by keyboard, and the latter for special names of mathematical functions such as sin, cos, log and so on. While for every MathSymbol object, our tool prepares an input button in the editing window, MathString objects are input by replacing the MathInput object created for incoming plain texts. In order to display MathSymbol objects, we define an object *Symbol* which performs either method of rendering, and attach its occurrence to each MathSymbol object.

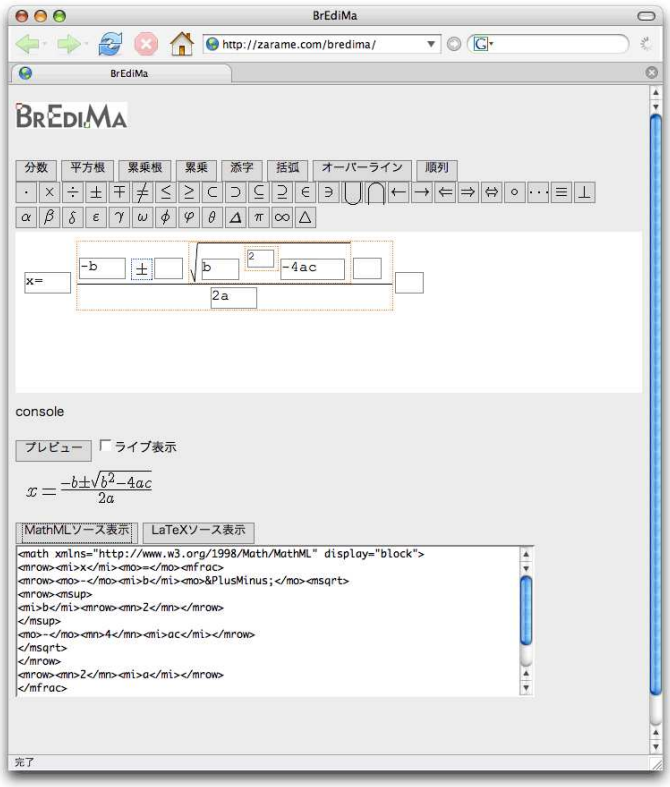
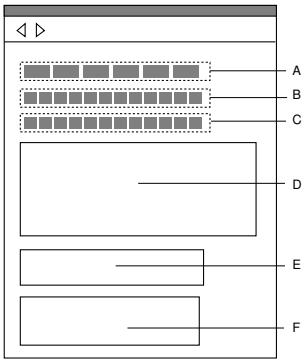


Fig. 1. Screen shot of the editor.

3.2 Display Layout and Rendering

Figure 1 is a screen shot of our editor. The screen is organized as follows.



A. buttons for container objects
B. buttons for token objects of operators
C. buttons for token objects of identifiers/symbols
D. workarea for input mathematical expressions
E. preview area: MathML or image generated by mimeTeX
F. output in MathML or LaTeX

Input and editing is done in the workarea in the middle of the screen. Each object occupies its own rectangular area in it, and is displayed by arranging its descendant objects inside the rectangular area recursively following the object hierarchy.

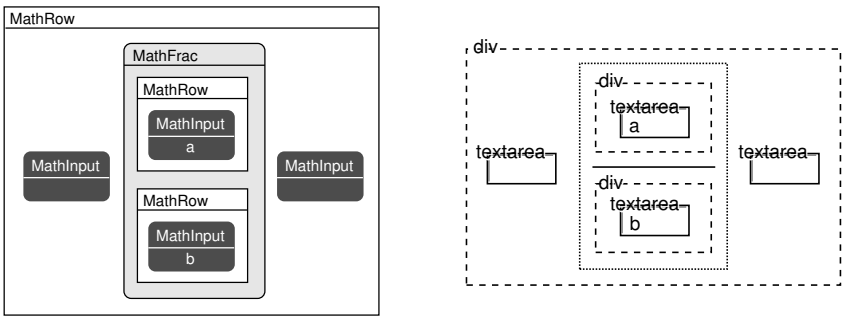


Fig. 2. The structures of objects and HTML elements of $\frac{a}{b}$.

Each Container object generates a `DIV` element as its own area in an HTML document, and arranges the elements of its children inside the `DIV`. The arrangement depends on the kind of Container object. For example, Container object of rational expressions arrange two sets of elements contained in the MathRow objects of numerator and denominator aligned vertically, and that of square root places the elements of the MathRow object on the right of a radical sign. The rectangular area of Container object is surrounded by dotted lines (Figure 3) to indicate the area and the contained objects, which will be helpful in editing. Each MathRow object also generates a `DIV` element, and arranges the elements

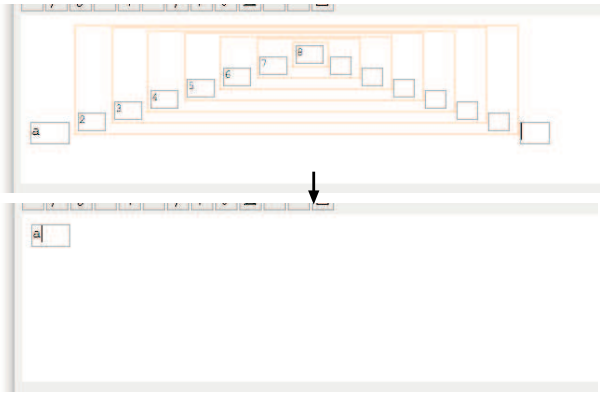


Fig. 3. Deletion of a Container object having multi-level descendant.

of its children inside the `DIV` from left to right.

Notice that inside the workarea, MathInput objects and Token objects are rendered in two font sizes. The style of the expression being edited is expected as similar to the formatted image as possible, however, too-small images are hard to recognize and edit. Also, we rarely need multiple sizes because expressions we treat are limited to those appear in high school text in our implementation. We decided to use two sizes. Compare the visibility and the understandability of a^{b^c}, a^{b^c} and a^{b^c}, and we will see that our choice is appropriate. In this situation, vertical positioning plays an important role for understandability. We compute the heights of the expressions above and below the level, and arrange the expressions in a MathRow object so as to horizontally align their centers.

3.3 Input and Insertion and Deletion of Objects

For MathInput, we use `textarea` element of HTML instead of a usual input method `<input type="text">`, in order to maintain the cursor position without being disturbed by focus change. We suppress the emerging of scroll bar for `textarea`, by the help of CSS.

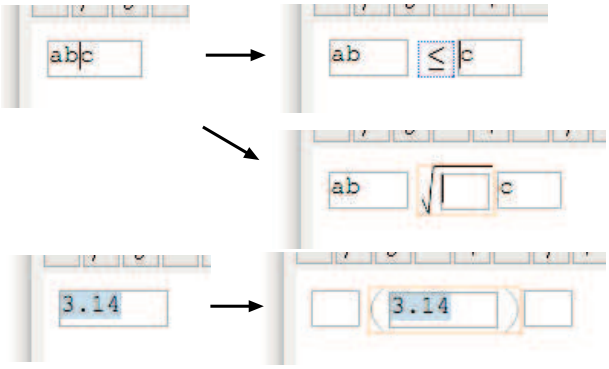


Fig. 4. Insertion of a Token object (top), a Container object (middle) and bracket-pair object for a selected text (bottom).

A new object is inserted by clicking an object button or by typing a special character at cursor's location. Then, the text string of the MathInput object on cursor is partitioned and a new object is inserted between the parts, as shown in Figure 4, which is performed by an `insert()` method of the MathInput object.

For every Container object and for every Token object, we put MathInput objects on the left and right, so as to enable insertion at arbitrary places in the expression under editing. This flexibility may seem inconsistent with structured editing, but is important to ease editing and to attain higher usability. Math-

Input objects serve to showing possible places for insertion, not merely as place holders for input.

Cursor Movement The above flexibility slightly complicates the control of cursor movement. Outbound cursor movement from a MathInput object by cursor (arrow) key invokes either method, `fwdCursor()` or `backCursor()`, of the Container object of the parent. This invocation is transmitted to the ancestors until an appropriate neighboring MathInput is found in the Container object, in which case the method `setCursor(0)` of the object is invoked to set a new cursor position. Backspace at the left edge of the textarea of a MathInput object deletes an object on the left. This is done by invoking the method `del()` of the parent MathRow object, which will call the method `removeObj()` of the target object. If the target is a Container object, all the contained objects are deleted, as show in Figure 3.

Adjustment of Object Layout Every time the content changes, the current MathInput object adjusts the size and the alignment and redraws the content to get a new layout. This adjustment is repeated for all its ancestors, until the outermost object is reached. This can be done by calling a method `layout()` which performs the adjustment of its own object and then calls the method `layout()` of its parent. Figure 5 depicts this process of adjustment; (A) two '0's are added in the denominator, (B) the method of the Container object of the rational expression realigns the numerator and the denominator and change the size of the display, and (C) the size change in (B) affects the positions of the objects on the right.

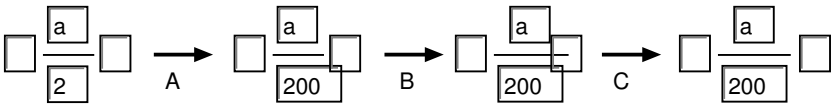


Fig. 5. Process to adjust object layout.

4 Application – A Simple Extension for Wiki plugin

As an application example to experiment the generality and the adaptability, we developed a plugin for Wiki to support input of mathematical formulas by utilizing our tool. The Wiki clone of our target is FreeStyleWiki, which equips with a LaTeX plugin and seems easy to introduce new plugins because highly moduled.

The plugin we implemented consists of two major parts. The one is a button of 'input of mathematical expressions' on Wiki's editing page to initiate our tool

with opening a new window, and the other is a button of 'return expression and quit' on our tool to exit from the running tool. The expression returned is in LaTeX adjusted to the LaTeX plugin, and inserted at cursor's position on the Wiki page for editing. The plugin in a pure sense that it extends the action of a Wiki program consists only of the addition of the button, whose unique code piece amounts to 10 lines at most. Also, the modification of our tool is done with several lines, and is almost independent of the types of Wiki.

Totally, the amount of code implemented is quite limited, and its Wiki-dependent part is very small. From our experiment, it turned out that our tool can be adapted with a small amount of change to software tools, including Web tools, with similar properties.

5 Summary and Future Works

We implemented an initial version of a tool for input and editing mathematical expressions. It is characterized by the use of only JavaScript in object-oriented programming, and the alternative use of MathML and mimeTeX for image generation. All editing and input functions, except the image generation by LaTeX, are implemented by JavaScript and will be executed on a client side. Therefore, it is expected to be used in various scenes of Web services. Actually, we implemented a plugin for Wiki, with a very small amount of effort. This experiment of the extension for plugin indicates that out tool has high transportability and generality required for Web tools.

The implementation is experimental and insufficient as an editor, although sufficient as an input tool. There are many functionalities of editing not implemented in our initial and experimental version, which includes saving to and reloading from files (appropriate file format must be fixed), undo, and so on. They are left for future study and development, as well as some extension using AJAX.

References

[1] mimeTeX manual. John Forkosh Associates, Inc. `http://www.forkosh.com/mimetex.html`.

[2] K.Nakamura. imgTeX. `http://www.eaflux.com/imgtex/index.html.en`.

[3] Texvc. Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Texvc`.

[4] mozilla.org. JavaScript. `http://www.mozilla.org/js/`

[5] J.J.Garret. Ajax: A New Approach to Web Applications. `http://www.adaptivepath.com/publications/essays/archives/000385.php`

[6] Document Object Model (DOM). W3C. `http://www.w3.org/DOM/`