

Boxes, Inkwells, Speech and Formulas

DRAFT

Richard J. Fateman

March 10, 2006

1 Introduction

This paper sets out some designs for entering mathematical formulas into a computer system. An initial approach to this task suggests that the previous model, namely writing mathematics on paper or chalkboard, should lead to a natural computer system using a stylus for writing on a tablet.

For feedback and for presentation such as in this paper, we use the typesetting capabilities of Knuth's \TeX system to show how “properly typeset” expressions might appear. We use \TeX here to show our design for an interactive input scheme, under implementation. For this to work, an interactive system must make expressions appear approximately in the same sequences as illustrated, on a computer display. The solid color boxes that appear in the incomplete forms are intended as invitations for the user to continue writing out a formula, continuing from within one of those boxes. Think of them as “virtual inkwells.”

For example, an attempt to write a superscript must begin by dipping the stylus (or mouse) in the superscript inkwell. An attempt to write an operand adjacent to an existing must begin in that inkwell. Initiating writing elsewhere on the screen will have no proper ink and will not contribute to the formula entry. We also point out that speaking the terms, rather than writing them, may provide more accurate communication.

At this point we suggest you look ahead a page or two to see some pictures of inkwells.

1.1 Why Inkwells?

This ink-well-based constrained input provides an obvious basis for cooperation between the human entering a formula and the supporting computer program. The program promises to try to parse the formula into something meaningful; the user must be able to express the intent of the writing using the provided paths that can be interpreted successfully. Experience with other programs written without such constraints [8], but using a free-format input demonstrates substantial problems. The user (especially one who is unfamiliar with or uncomfortable using a stylus or mouse) will generally not be able to write glyphs of the “proper” size, and will not precisely position them. Other programs [7] repeatedly analyze the sequence of strokes to find new and possibly better interpretations of the markings. This reinterpretation may convert correctly-recognized symbols into incorrect ones. On the other hand, since the input is unstructured, one can go back and insert extra parentheses or other markers as necessary.

We make a simplifying assumption that the input will always proceed roughly left-to-right, with occasional diversions up and down. The appearance of inkwells will force this direction.

The strategy for the stylus once it has been inked, is critical.

- The user is not constrained to keep the stroke inside the inkwell. Once there is ink, a lower-case descender letter like “p” would be expected to drop below the inkwell.

- The user does not have to write small letters: once the stylus has ink, it can be used to write any size characters. (We could also zoom in on the expression.)
- The treatment of second and subsequent strokes can be unified in two ways:
 - If they are drawn close together in time, they are part of the same character. Thus a multiple-stroke K or E would have to be written so that the interval between the strokes is brief.
 - A new (large) drawing pad can be displayed so that anything within that pad is part of the same collection. This would allow a user to write more extensive content for a box. Instead of writing $ax + b$ in 4 separate steps, it could be written in one. The Microsoft writing-pad application which can be docked anywhere on the screen is attractive to serious users. The alternative “write anywhere” design appears to be trickier to use, but this is a human-factors issue needing study.
- As indicated below, speech input may be appropriate. If we can accurately recognize spoken “eks squared sign squared of eks” as $x^2 \sin^2 x$ then we have a potential for rapid input. Consider the symbol $\not\subseteq$ which is unicode (hex) 2288. You might speak it as “neither a subset of nor equal to” [3], but reliable recognition of the handwritten glyph, given the number of similar symbols, would be pretty difficult.

The sizes and spacing of some objects in the formulas will have to adapt as the formula is written. These growing objects include divide bars, parentheses and other brackets, matrix delimiters, integral signs, products, summations. The size of variables will depend *not on how large a user writes them*, but their positions relative to other symbols. The rules for T_EX provide for careful evaluation of appropriate spacing and size, with the possibility of user-specified exceptions. The context (display or in-line) as well as global directives, can also affect size. A functional description of T_EX layout [4] may be more useful a guide as to how to re-implement this than the original description or code.

1.2 Inkwells may be speech cues

It is not required to enter material in these boxes by writing. Once a position is selected, it may be preferable to *speak* the glyph. As an example, a vertical bar may be nearly the same as a slash mark “/” or the number 1, or a number of other glyphs that look similar. A vertical stroke may also be part of any number of multi-stroke symbols such as K or P. A horizontal line might, among many possibilities, be “minus” or “divides,” or part of “=”.

Speaking removes some ambiguity. It may also be faster as a consequence. Saying *bold small tee* to produce **t** may be quicker and more accurate than trying to write a symbol which might easily be confused with +. There seems to be no intuitive “embolden” written notation. (In the past, instructions given to an old-fashioned human typesetter tended to require underlining with color pencils). A simple speech application that allows recognition of symbols spoken as (for example) “Greek capital gamma” seems to be quite accurate, at least for a small set of alternatives.

Once one begins to use special symbols, the need for disambiguation of apparently similar glyphs increases.

A handwriting recognizer asked to distinguish similar isolated characters would not be reliable, and even if one could be constructed to make such distinctions, it would require an exceptionally careful human writer: \langle , $<$, and \angle , L and even α . Whereas *left angle bracket*, *less than*, and *angle*, to use the T_EX names, are rather different spoken utterances. There are thousands of spoken symbols that could be taught to the computer system. Names need not be unique; any of several common names might be used for the same glyph. The human writer would not have to signify fonts by writing in *Fraktur* or *Blackboard* glyphs.

A combination of speech and handwriting may be most helpful. Our working hypothesis at this moment is that in a multimodal input method based on ink-well constraints relies primarily on position information for which we can use a mouse. The handwriting recognition can be down-played if the symbol specification

burden is (mostly) carried by speech. We wonder how natural this might be to someone who comes to this task with a preconception that “mathematics is normally handwritten”.

The listing of MathML entities running from variations on the letter “A” through “greater-than-or-equal, slanted, dot inside” to “script letter z” has 1293 glyphs. This does not however include all font variations or modifiers such as bold, underline, etc. It does not correspond directly to T_EX notation, but the latter is extensible through font-construction programs.

1.3 Checking Display Preliminaries

In order for you to read this paper, we must first be assured that the system that is being used to view or print it displays the right size boxes in our T_EX macros relative to the usual typeset material. The red rectangles below should approximate the sizes of the A and the x . (The same macro is used for each rectangle, but the expansion of the macro depends on the context within the T_EX expression).

$$A \color{red}{\square} x \color{red}{\square} X$$

Here are some samples showing expressions and their transcriptions with pieces replaced by rectangles. The geometrical arrangements should be similar.

$$A_x^y B = \color{red}{\square} \color{red}{\square}$$

$$X_A^B C = X \color{red}{\square} \color{red}{\square}$$

$$\frac{x^y + z}{A + B} = \frac{\color{red}{\square} + \color{red}{\square}}{\color{red}{\square} + \color{red}{\square}}$$

1.4 A typeset example

In order to be concrete, we will use a running example for several sections. Here is one sample expression, displayed as T_EX typesets it:

$$x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{1}{x} dx$$

2 Constrain by Leftmost Entry of Expressions

The principal rule we need to provide for our users is that whenever possible expressions should be written from left to right. This is the normal direction, but occasionally one makes mistakes. We allow the user to select an expression previously entered, and “write over it” perhaps with a much more elaborate expression, but not all previously written sub-expressions can be selected (for example, in a sum of five items, select the middle three.)

2.1 Templates for Super/Subscript and Right

Let us begin the entry of a particular expression, in pieces, using our first design. The general expression template allows for a superscript and a subscript, and a position to its immediate right. Only rarely are all three filled, so this may seem like overkill for providing templates, and to the user both complicated and clumsy.

$$x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{1}{x} dx$$

We continue in this way to eventually reach:

$$x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{1}{x} dx$$

The only way we could make this sequence even more cluttered might be to allow expressions to grow to the left, above, below, as well as pre-sub and pre-superscripts.

2.2 Clearing some clutter

Here is another design. We leave off the subscript template boxes and require that a writer who needs to produce x_2 writes simply x and later goes back (e.g. click with right mouse button within the glyph bounding box) to choose from a menu: {delete, copy, edit, subscript, font, size}. Choosing subscript then opens up a subscript template. A selection method (where exactly does one point?), as well as appropriate menu items for *compound* expressions requires careful design. One method we have used requires selecting $\alpha + \beta$ by clicking inside the box for the expression, but outside the boxes for the three components. Another method uses a rectangular lasso to contain some or all of the selection. The menu available for a selection depends on what is selected: an integral sign would have different menu properties from symbolic atoms, or from the sum of two terms.

It seems that a programming “type” structure would be almost inevitable, at least for most sophisticated users. We could declare (within a delimited sub-scope of the formula entry program) that all use of specific symbols have special attributes. For example we could declare that A should always be set in bold as **A**, the sequence of two letters sn should be set in Roman as *sn*, and symbols otherwise not in the handwriting lexicon (or difficult to distinguish from other symbols) should be entered in distinct forms. That is, one might really want to write $A \vee B$ as **A vee B**, since the letter *v* (handwritten) is not distinct from \vee . One could also specify temporary shorthands like **D** mapping to $b^2 - 4ac$, or long-hands like **dot** to represent the period, or **cdots** to represent three centered dots \cdots each of which would otherwise be hard to recognize in handwritten form. (See section on speech for more discussion on this ambiguity.)

Returning to our constraint-box design, if we eliminate the subscript template box by default, our example above becomes simpler to view. Introducing subscripts then requires the user to go to an edit menu to make the x into an x_2 or use some kind of macro-expansion that makes the utterance $xsub2$ into a subscripted x .) Here’s how it might look:

x 


Edit (using a method not shown here) the x to make it x_2 , either now or later.


x_2 


x_2^3 


At the point where we enter the “+”, the system has only one extraneous box, and then it has no extraneous boxes at all.

$x_2^3 +$ 

$x_2^3 + \sqrt$ 

$x_2^3 + \sqrt{y}$ 

$x_2^3 + \sqrt{y^2}$ 

$x_2^3 + \sqrt{y^2 +}$ 

$$\begin{aligned}
& x_2^3 + \sqrt{y^2 + z^2} \blacksquare \blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} \blacksquare \blacksquare \blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int \blacksquare d\blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int_1^{4\blacksquare} \blacksquare d\blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int_1^{4\blacksquare 5} \blacksquare d\blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{\blacksquare}{\blacksquare} d\blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{1}{\blacksquare} d\blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{1}{x} d\blacksquare \\
& x_2^3 + \sqrt{y^2 + z^2} + \int_1^{45} \frac{1}{x} dx.
\end{aligned}$$

Comparison of the entire sequence makes it even clearer that removing the subscripts from most templates makes the overall construction more comprehensible.

2.3 Speech, again

Remember our suggestion for speech input: the symbol \vee above could be written as vee but perhaps more easily spoken as “logical or.” We have just begun to experiment on the hypothesis that speech input *generally* to the computer could actually remove the need for some of these “macro” commands and also make the symbol recognition far more accurate. The point is particularly clear when the handwritten glyphs are problematical. Speaking “dot” may be a key to recognizing what might otherwise be a (small) piece of essentially random ink.

2.4 Other improvements

Another possibility is to use color coding to keep track of what is likely to be the main course (most likely continuation) of the expression. Thus we could illustrate that we expect that the user to use (one of) the blue boxes next. We could even advance the point of insertion to the blue box so that a continuation of speaking or typing would proceed automatically to the next blue box. Or voice commands could navigate among the boxes.

$$\begin{aligned}
& x \blacksquare \\
& x^3 \blacksquare \blacksquare \\
& x^3 \blacksquare \blacksquare \blacksquare \\
& x^{34} + \blacksquare
\end{aligned}$$

$$\dots$$

$$x^{34} + x \sin x$$

In this example, speaking “eks up three four down plus eks sign eks” would be acceptable.

There are a host of features that may make sense, especially for serious users, beyond simple handwriting and speech.

Here are some options:

- We could ask users up front: Do you expect to enter any subscripted variables? If so, what are their names? Do they have subscripts or other marks? (This context and other similar information below could be saved, restored, edited, etc.)
- We could ask about unusual notations, especially those that might be easily confused in handwriting like $\langle \rangle$ versus $\langle \rangle$? How would you prefer to notate these? How do you plan to speak their names? Are they constrained as in the typical bracketing of parentheses?
- We should provide a method for adding new templates, such as a template that starts out with equation numbers on the left

(■) ■

. Another template that might be used in tables of integrals, would have spaces for side-conditions restricting the domain of some of the extra parameters, cross reference to related formulas, and even a reference to the literature.

- Once an expression is entered (or computed), there is a potential need for alterations. The design for such editing is not covered in this document. We should explore this further [2].
- It should be possible to scroll a large expression, zoom in or out of expressions for ease in writing or reading, and it should be possible to fold expressions over more than one line. This last issue is considerably more difficult to address adequately, and may be best ignored for lack of a good automated solution. A good multi-line expression could be entered on several lines or in a matrix by a skilled typesetter.
- We have written as though a document consists entirely of a single formula. This is, of course, unlikely². A useful entry system should allow interspersed ordinary text both in-line and in complete paragraphs surrounding displayed mathematics. More specialized combinations of text and formulas as in theorems or proofs should be accommodated.

2.5 How much can be written in one box?

It is clear that the tiny boxes are insufficient space to write characters; that is why we refer to them as inkwell. But it would be convenient to insert many characters in a box: in fact, as many as can be conveniently written or spoken. Thus $e^{\sin ax}$ would most plausibly be written in only two boxes, at least if the whole exponent can be written before the ink dries. One technique used in Microsoft Office’s handwriting interface is to provide a separate writing pad, as opposed to the “write anywhere” mode. (The major recognition advantage is that the user/computer have a baseline on which to write/recognize. The major disadvantage is that you probably want to use the stylus as a pseudo-mouse, too. Dual use is confusing.) In our case a writing pad could be implemented by a local pop-up window pad which allows more written material. How large should this pad be? (Should it grow?) How long should it last before the ink dries? Should it *always* appear?

²Except in spy novels where the formula is secret.

These are delicate issues – but resolving them may not be important. In our experience, a handwritten version of $a + b \sin \theta$ is hard to recognize regardless of where it is written. A result of “at b sino” is typical. Even the untrained speech program in Microsoft Office comes out pretty close: “a+be sign data”. A more restricted speech grammar we are developing (Fall, 2003) to demonstrate applications of the Speech SDK seems to work rather more accurately, given a subset of words or phrases typical of mathematics, including standard alphabetic characters.

3 More examples

It is tempting to provide special treatment for “known” functions. That is, we can assert that uttering the symbol \sin immediately produces a template which provides for three new boxes: the “power” of the \sin , its argument, and the (optional) continuation. That is one could create $\sin^2(xy)$ by starting with this template:

$$\sin^{\blacksquare}(\blacksquare)\blacksquare.$$

This template suggests (a) the parentheses are not optional. (b) The next datum to be entered should be the argument of the \sin . (c) The power and the continuation to the right are optional or secondary after the argument is filled in.

The downside to this is that it becomes impossible to typeset $\sin x$ without parentheses. Whether this is an issue or not is a matter of taste, about which it is difficult to legislate.

A complete implementation should provide display of glyphs that grow, such as brackets and matrices. See how TeX handles them:

$$\left(\frac{(a+b)c+d}{e^{1/(1+x)}} \right) + \sin(x+y)$$

We can address the need of someone who writes $a + bc + d$ and subsequently decides to enclose this expression in parentheses as $(a+b)c+d$ by editing operations, since it would otherwise violate our procedural guidelines of starting from the left and proceeding to the right. One technique is essentially to use another piece of canvas to write a \blacksquare , then select the “inside” $a + b$ out of an existing form, and past it into the \blacksquare . The complete scenario is thus more elaborate than just overwriting an expression with some parentheses.

4 Comparison with Other Programs

The lengthy “A Survey of User Interfaces for Computer Algebra Systems” [6] provides a broad perspective on the topic prior to about 1998. The historical highpoints include, in our view (R. Anderson’s early thesis, W. Martin’s program, MS equation editor, recent tour-de-force programs such as Milo/PacificTech, Expressionist/Theorist/Livemath, and Soiffer’s work in Mathematica). Recent progress in handwriting or speech input is not included.

4.1 Keyboard input

It is hard for us to be critical of keyboard input of mathematics: we have been using this method for many years, and there are several programs that provide excellent support for the higher levels of expertise. We first point out some, perhaps obvious, points against the keyboard for math input.

- Math is inherently “locally two-dimensional” as x^2 and therefore is a mismatch to the unadorned keyboard, which provides a “locally linear” string. By this we mean that the two-dimensional page layout of text is done by a global operation of inserting “new line” characters.
- Given that a pen or a mouse is used for geometric positioning, forcing the user to switch to the keyboard to provide the content is inconvenient.

In favor of keyboards, we have these considerations:

- Handwriting (or speech) is not always accurate and efficient. If errors cannot be handled in the same mode, the user may be forced to learn about keyboard entry in order to use it for corrections.
- Once the keyboard is introduced either for content or correction, the design can easily go a step further and use it (via “arrow” keys or other codes) as a replacement for the mouse/stylus selection. This is a possibility: using the *tab* key to circulate sequentially among possible selectable boxes is a convention with filling in HTML forms. One program (Mathematica) uses `control+space` to drop “out” of a construction; TeXmacs uses the *rarrow* key similarly. With such a convention and a few clues as to how to navigate around a matrix or a \sum , a skilled typist can do very well in positioning. Making use of macro definitions for common subexpressions or frameworks, a common tool for the expert in T_EX can provide even more efficiency.

We mention a few examples of keyboard input (although the [6] provides much more). TeXmacs is a free open-source editing system [5], inspired by the `emacs` editor and T_EX. It is not implemented using either one, however. TeXmacs (once it is successfully installed) allows you to interactively type (and see as immediate feedback) almost anything in T_EX, as well as various extensions to images or other objects. TeXmacs allows alternative methods. For example you can choose from a menu a fraction *template* or alternatively type `\frac` in place or the shortcut `jem alt-fi`, to then be shown an in-place template for numerator and denominator. The non-menu approach is preferable since you can keep your fingers on the keyboard. (TeXmacs has keyboard shortcuts, but this could presumably be added.) To change from text to math and enter a fraction $\frac{a}{b}$ you type 8 characters: `$ alt-f return a down-arrow b right-arrow $`. TeXmacs uses font derived from the excellent T_EX repertoire. Occasionally (most often when first starting up with a new installation) it pauses to compute new font sizes as they are needed for the first time. TeXmacs is a highly competent text typesetting system, but as a math system it is designed to interface with other systems, and does quite well ordinarily. There are issues with conflicting system constraints on interfaces, handling errors, etc.

One of the more popular computer algebra systems, Mathematica [9], provides for keyboard/mouse input. The standard input is a linear keyboard input format, but recent versions of the system have added 2-D input formatting which can be accessed by menu or by keyboard escapes. The running WYSIWYG input/output is not as elegant in appearance as the TeXmacs system, but is similar in that it refers back to a similar repository of mathematical symbols from T_EX. Mathematica generally assumes you are in “math” mode so to enter a fraction $\frac{a}{b}$ you can type *a control-/ b control-space*.

Command completion is available in either systems so you can avoid typing some long names. In each of these systems you can move a cursor to the left into expressions. TeXmacs provides both visual feedback (boxes) as well as a brief written description in the lower right corner of your location in an expression (e.g. hints of the containing operators of the selection point).

The Mathematica program places some demands on the input expressions that are not entirely commonplace, such as enclosing function arguments in square brackets, and using a special “d” for the differential marker in integrals. (It is possible to type material in “Traditional” form with conventional appearance: this can sometimes be properly interpreted and edited, but frankly traditional form is ambiguous: $a(r + s)$ means one thing if a is a function, but if a is a constant it indicates invisible multiplication). Expressions entered into computer algebra systems like Mathematica tend to get simplified and rearranged, sometimes to a disconcertingly different form, and sometimes to great benefit.

The TeXmacs program is generally less concerned about the meaning of expressions than Mathematica because TeXmacs does not have to fit every utterance into a particular computer algebra model of representation and evaluation. This can be used to advantage when especially when defining new notations.

Certainly considering the price, TeXmacs deserves careful scrutiny as a mathematics entry program. For Windows users, TeXmacs currently requires an installation of Cygwin, and in my experience takes some fiddling to get to work. It can export files as T_EX although that is not the native form for TeXmacs.

We should also mention the equation-entry keyboard system that probably requires minimal fiddling to install (for Microsoft Word/Office users): the Microsoft equation editor and its upgrade to MathType (www.mathtype.com) from Design Science). This program is capable of good typeset results, although it too seems to be difficult to use without interspersing significant mouse activity between keystrokes. The design is not any more keyboard-hostile than others: many of the menu choices can be reached by combination keystrokes or reached by navigating menus by arrows.

Every program requires some effort for odd symbols; the palettes may be extensible, and as unicode becomes more easily available, the symbol set can presumably have thousands of glyphs in different fonts and sizes.

Conclusion: We can't count out the keyboard as an important input device, especially for people who are using advanced notations and can type accurately. Furthermore, it is likely that unusual notations and glyphs will be introduced first via keyboard input. By the same token, the steep initial learning curve is a barrier for naive users.

4.2 Other Math+Speech Efforts

What appears to be a primitive version of speaking math is embodied in a program Mathtalk <http://www.metroplexvoice.com/> using a version of the software Dragon Naturally Speaking. The demonstrations are not encouraging, but the fact that it works with MacKichan Scientific Notebook is promising. (A discouraging aspect is the suggestion in the demos that a military-style alphabet is used: thus dx might be spoken as "delta x-ray". It is not clear if this is essential. In our own experiments, single-letter recognition is somewhat unreliable with the letters {b, d, t, p} sometimes confused. Recognition of phrases like "d y by d x" is far more reliable.)

5 Conclusions and Further Work

We have suggested a novel handwritten technique which displays all possible continuations of a formula, writing it left to right. The user is constrained to draw "ink" from these continuation spots. This feedback promotes the relatively painless construction of well-formed formulas.

Secondly, we have suggested that, as a minimal effort to take advantage of multimodal input, individual symbols or short phrases inserted by speech input can be synchronized with handwriting or pointing. It may in fact be easier to *replace* the entering of most handwritten symbols by speaking them.

Other uses of speech might include using voice commands to edit a symbol or make an alternative recognition choice, or to give higher-level commands like "begin boldface".

We completed a demonstratin prototype; actually two.) The first by Kevin Lin (SKEME) demonstrates the boxing and display, and is written in Lisp. The stubs for speech and handwriting were not fully developed. A system called Math Speak and Write, with a somewhat less evolved display, but one linked to both speech and handwriting was written by three students, is of sufficient robustness that we offer a download of it, <http://www.cs.berkeley.edu/~fateman/msw/mws>.

Continued improvement of this system leads us to go back to basics and find a way to recognize math symbols without extensive training on a per-user basis, and also voice recognition, with error correction, using a context-free grammar that describes math. We hope to be able to test our design against other systems [1, 8, 7, 5] for use in two areas: producing documents, and as an alternative (to typing) as a mode of input for computer algebra systems.

Finally, we feel it appropriate to point out that even recognition of forms syntactically, and reducing them to an apparently unambiguous linear form does not imply that the recognition program has a complete semantic model of the utterance. The meaning behind a notation is necessarily context dependent. One of my favorite examples occurs in the use of dx in a published integral table entry for the formula $\int (a+bx)/(c+dx)dx$.

6 Acknowledgments

This research was supported in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley. Student assistants were supported in part by an Undergraduate Research Opportunities grant from the UC College of Engineering and the Haas Scholars fund.

We are aware of the irony that a Berkeley project should advocate constrained speech input, as opposed to free speech.

References

- [1] J. Arvo, *FFES: Freehand Formula Entry System*. <http://www.cs.queensu.ca/drl/ffes/>.
- [2] S. Dooley. “Editing mathematical content and presentation markup in interactive mathematical documents,” *Proc. ISSAC 2002*, (Lille, France) 55–62. <http://www.mathmlconference.org/2002/presentations/dooley.xml/>
- [3] R. Fateman. “How can we speak math?”
- [4] R. Heckman and Reinhard Wilhelm. “A Functional Description of $\text{T}_{\text{E}}\text{X}$ ’s Formula Layout,” *J. Functional Programming* 7(5), 451–485 (1997) <http://rw4.cs.uni-sb.de/heckmann/doc.html>
- [5] Joris van der Hoeven. *TeXmacs* (<http://www.texmacs.org>).
- [6] Norbert Kajler, Neil Soiffer. “A Survey of User Interfaces for Computer Algebra Systems.” *J. Symbolic Computing* 25(2): 127–159 (1998)
- [7] N. E. Matsakis. Recognition of Handwritten Mathematical Expressions, *MS report*, MIT <http://www.ai.mit.edu/viola/research/publications/matsakis-MS-99.pdf> (1999) also demo at <http://www.ai.mit.edu/projects/natural-log/>
- [8] M. Suzuki, *Infty Project*. <http://www.math.kyushu-u.ac.jp/suzuki/index-e.html> (includes links to download of demo)
- [9] S. Wolfram et. al. *Mathematica*, <http://www.wri.com>.