Towards a Multimodal Interface for Mathematical Manipulation

Kyle Miller
kmill@mit.edu

29 January 2010

Abstract

This paper presents elements to a possible symmetric multimodal user interface for manipulating mathematical expressions. We will discuss multimodal user interfaces, what subset of mathematics is tractable, previous work in this area, novel ideas to progress such multimodal interfaces, and a direction for future work.

1 Introduction

Mathematics is the study of the symmetries and structure in abstract objects such as numbers and geometric figures by using strictly logical arguments. And, as such, it is a very broad subject. There are many subfields of study such as geometry, analysis, and algebra which focus on shapes, change, and structure, respectively. The distinction between these subfields is not clear-cut, however, as they share techniques and methods which mathematicians have found useful.

An important part of mathematics is to communicate results to other mathematicians. For this, a rich language of words, figures, and symbols has been devised, each with a precise definition.

The following is a scene to motivate a multimodal user interface.

1.1 Painting a Scene

Two colleagues are standing before a chalkboard discussing a mathematical idea, the roots of a quadratic polynomial (these colleagues are not particularly advanced in mathematical studies). The first colleague writes the definition of a quadratic equation, $0 = ax^2 + bx + c$, draws a picture of a general parabola, showing the relationship between the roots and the intersections on the x-axis, and begins to complete the square to derive the quadratic formula.

Meanwhile, the second colleague is asking questions to clarify—"Is that symbol supposed to be an x?", "Is that a parabola?", or even "What happens if the parabola does not intersect the x-axis?" if the second person is bright.

The second may also take the chalk to help with the derivation or to draw a picture or an equation to aid in asking for clarification. Both speaking and drawing are symmetric since each of the colleagues are able to use these media for clarification.

1.2 Symmetric Multimodal User Interfaces

Between a user and a computer exists a user interface, which is the protocol with which the user manipulates the computer and from which the user is supplied information. From the point of view of the computer, this is a sort of API to the real world.

There are many modes of input a user interface can use. Commonplace are keyboards and mice, which allow a user to input characters and point at things, respectively. Rarer are microphones for voice input, pen digitizers for writing and drawing, video cameras for gesture input, or electroencephalograms for controlling things via brain waves.

Also, there are various modes for supplying information to a user. For instance, a computer monitor or projector gives visual information, speakers give audio feedback, or force-feedback joysticks can supply a sense of touch.

A multimodal user interface is a user interface which blends together many input and output modes. The advantage of a multimodal user interface, if well designed, is that the weakness of one mode can be offset by the strengths of another. One example is the combination of a keyboard and mouse. A mouse lets one navigate around a two-dimensional on-screen interface of widgets. But, being like a single finger, it is not suited for text entry. However, this weakness can be augmented by a keyboard, which people have used for the last hundred years for speedy text entry, but which are not well suited for selecting on-screen widgets.

Well designed multimodal user interfaces are still use-

ful if a user chooses to use any subset of modalities. For instance, a cell phone with both buttons and speech recognition should let one use either only the buttons (if the room is noisy), only the speech recognition (if one's hands are busy), or both at once. However, for a natural interface, speech shouldn't simply be an equivalent to the button input, with voice commands such as "move cursor up" or "type an 'a"'.

A symmetric multimodal user interface is a user interface in which the computer can respond to the user using similar modalities. More interesting are symmetric multimodal user interfaces which also allow for a dynamic dialogue.

An example of this is a program by Aaron Adler under Professor Randall Davis which had the user draw and talk about a diagram representing a physical situation, such as a ball above a lever, and the computer would ask questions about what would happen, also drawing on the diagram for clarification. Part of the user interface is an artificial intelligence which knows a little about physical systems that can maintain a coherent dialogue.

1.3 Analyzing the Scene

Between the colleagues there is a chalkboard, and they are able to talk to each other. There is a symmetric, multimodal interface between the colleagues whose modes are drawing and voice.

2 Mathematical Interactions

The goal of a symmetric multimodal user interface for mathematics is ultimately to be able to replace one of the two colleagues with a computer in a seamless manner.

This is a lofty goal, however, as this is equivalent to passing the Turing test. A more reachable goal is to take some aspect of the interaction between the colleagues and make a program which can simulate that.

Here are a few kinds of general interactions one may find between two people during a mathematical discussion:

- Creating a relation between variables (using an equation)
- Drawing a diagram or giving an example of an object
- Asking to generalize a concept
- Defining a new word
- Proving the validity of a statement
- Finding a property of an object

Examples of these, although less friendly than a real person, can be found in mathematics textbooks.

One particularly useful technique of mathematics comes from algebra, and that is the manipulation of unknown quantities in equations (said another way, using a variable such as x). The behavior of these unknown quantities is well defined, and algebraic analysis has proven useful in fields beyond mathematics, such as physics and economics.

Due to the utility and defined nature of equations, a good first step in making a symmetric multimodal user interface for mathematics is to make such a system for the manipulation of mathematical equations. The user interface should support some of the interactions described for manipulating equations as objects.

3 Mathematical Expressions

Mathematicians have developed a fairly standard written language for describing the relations between quantities. The basic unit is an expression, which, when evaluated, has a value. Expressions can be joined together with operators or related to make equations or inequalities.

A basic expression is a variable, such as x or y, or a quantity, such as 5. Expressions may be combined using infix binary operators such as $+$, $-$, $\times$, and $\div$. There are also unary operators such as $\sqrt{}$. Higher-order operators exist such as $\frac{d}{dx}$ and $\int dx$.

There are two modalities for communicating mathematics: writing and speech.

3.1 Written Expressions

The recognition of written mathematics notation has been well studied over the course of the last 20 years. A survey of the field is in *[K. Chan and D. Yeung]*. In fact, Windows 7 ships with a mathematics notation recognizer. For these reasons, the problem of analyzing mathematics notation will not be discussed here.

3.2 Spoken Expressions

The written mathematical notation can, to a limited degree, be spoken. Mathematicians frequently speak simple expressions when conversing for clarification. Or, if a visual aid such as a blackboard or paper is not available, it sometimes becomes necessary to speak expressions.

Speech recognition of verbal mathematics has not been suitably researched. In fact, the author could only find

an application called MathTalk by Metroplex Voice Computing, Inc., which provided a verbal command-based language for constructing mathematical expressions on a screen. To communicate $\frac{1}{2} + x$, a user would speak "One over two. Move outside. Plus x."

4 Direction of Research

Since a symmetric multimodal interface ought to be able to use each of its modes, an expression manipulation interface ought to be able to use both written and spoken equations. Because written equations have been well studied, the direction of the research presented in this paper was to focus on spoken equations, and, for the purposes of maintaining a symmetric dialogue, for both recognition and synthesis. Some progress was made in understanding how equations are spoken and how to make a computer speak equations.

5 Understanding Spoken Expressions

First, to understand how equations are spoken, a list of operators and possible ways of reading the equations was compiled.

- $a + b$. "the sum of a and b", "a plus b", "a and b added [together]"
- $a - b$. "a minus b", "subtract b from a", "the difference of a and b"
- $a \cdot b$. "the product of a and b", "a times b", "a multiplied by b"
- $\frac{a}{b}$. "a divided by b", "a over b", "the quotient of a and b"
- $\sqrt{a}$. "the square root of a", "the root of a", "root a", "radical a"
- a^2. "a squared", "a to the second power", "a to the power of two"
- a^n. "a to the nth power", "a to the power of n"
- $f(x)$. "f of x", "f applied to x"
- $f(x) = a$. "f of x equals a", "f of x is a"
- $\sum_{i=a}^{b} c$. "the sum of c for i from a to b", "the sum of c for i equals a to b", "the sum for i equals a to b of c"

From this, an informal study was devised for which volunteers were asked to attempt to speak twenty-five expressions of varying difficulty. The expressions were chosen to determine how precedence of operators would affect the grammar chosen for the utterance.

1. $5 + x$
2. $2x + y$
3. $1 + xy$
4. $\frac{1}{2} + x + \frac{1}{y} + \frac{a}{b}$
5. $a(x^2 + 2)$
6. $(a + b)(c + d)$
7. $\frac{x}{2} + \frac{1}{3}y$
8. $\frac{x-1}{2}$
9. $\frac{x}{2-y}$
10. $\frac{x+y}{2-y} + \frac{z+2}{4z}$
11. $x + 2^{n+1}$
12. $x^{\frac{1}{2}} + x^{\frac{1}{3}} + x^{\frac{1}{4}} + x^{\frac{1}{5}}$
13. $e^x + \ln x + \cos y + \sin z + \tan^2 \theta + z \sin^3 \alpha$
14. $\log_2 (x + 2) + 7z$
15. $\lim_{h \to 0} (1 + h)^{\frac{1}{h}}$
16. $ax^5 + bx^4 + cx^3 + dx^2 + ex + f$
17. $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$
18. $\sum_{i=1}^{n} i^2$
19. $\frac{1}{1+n} + \sum_{i=1}^{n} \left(1 + 3i + \frac{n}{i^2}\right)$
20. $f(x) = \sqrt{1 + x^2}$
21. $2\sqrt{1 + f(1 - \sin x)} + 2$
22. $\frac{1-z}{\sqrt{f(xz)}}$
23. $\int_0^1 x^2 \, dx$
24. $\int_a^b \left(1 + \sqrt{1 + \left(\frac{df}{dx}\right)^2}\right) dx$
25. $y \in \{1, 2, 3, x, \sqrt{1 + z}, e, \pi, \epsilon, \delta\}$

There were seven people who volunteered, and their utterances were recorded. The volunteers commented that they thought the more complicated expressions were ambigous to say.

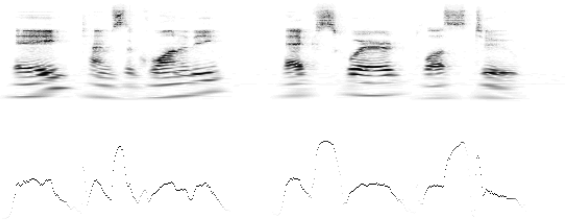

Figure 1: The utterance for $a(x^2 + 2)$, "a times the quantity x squared plus two," as a sonogram (above) and as a pitch representation (below).

5.1 Analysis

Analyzing the recordings, it was found that the volunteers made many mistakes, had many pronunciations for symbols (for instance, "z" was pronounced as "zed" by one volunteer), and generally chose the infix location for operators ("a plus b" instead of "the sum of a and b").

In speech, prosody is the patterns in pitch an stress. Interestingly, each volunteer tended to use similar pitch contours for the utterances.

5.1.1 Pitch Analysis

Speech, unlike the sound from a music instrument, does not have a definite pitch. Fundamentals can be missing, 's' sounds and other consonants are pitchless, and multiple pitches are sounded at once (for vowels). However, from listening to the recordings, it is clear that there is a general sense of pitch contour.

A novel way to analyze the pitch contour of a sample of sound is to first create a sonogram of the sound sample. A sonogram is a graph with time and frequency axes, and at each point, the amplitude of of the frequency at that particular time is recorded. The upper graphic of figure 1 is a sonogram where blackness corresponds to the magnitude.

Then, for each column of the sonogram, we imagine bar whose linear density is the magnitude at each frequency. We then find the center of mass of this bar. Or, stated differently, for frequency distribution at time t, $f_t(\omega)$, between frequencies a and b, we find the "pitch center" p_t:

$$p_t = \frac{\int_a^b x f_t(x)\,dx}{\int_a^b f_t(x)\,dx} \tag{1}$$

Then, we can also assign an "average energy" e_t to each time, which we define, simplistically, to be

$$e_t = \int_a^b f_t(x)\,dx \tag{2}$$

Then, we can create a new sonogram which only has pitch p_t with magnitude e_t at each time t. An example of this is in the lower graphic of figure 1.

To perform this analysis, the code reproduced in section 8.1 was developed. This code also performs a low-pass filter on the changes in p_t to produce a smoother pitch contour than that seen in figure 1.

An interesting feature to note is that any time an 's' sound occurs, p_t shoots up a few hundred cycles per second. These spikes should be ignored when looking at the pitch contour because they are not heard as pitches.

Also, when converting the sonogram back into a sound clip, the simplified sound turns out to sound remarkably like the original utterance.[1]

5.1.2 Pitch Synthesis

The reason for examining the problem of pitch synthesis is that, for a symmetric dialogue, it feels more natural if the computer can respond in a somewhat realistic manner, with some semblance of prosody.

After examining the pitch contours, a simple model for the pitch countour of utterances of mathematical expressions was developed. The general idea is that an expression as a unit has a global pitch contour to denote that the speaker is saying an expression. And, each subexpression also follows this some general contour. Then, the many levels of pitch contours are combined to make the contour of the entire utterance.

Also, there are pauses between subexpressions to help make the full expression clear. As can be seen in figure 1, the gap between "quantity" and "x" is much larger than the gap between "squared" and "plus." We model this as a doubling in the size of the gap between each subexpression with respect to the gap between each subsubexpression.

Pitch contours are modeled as piecewise linear functions from time to standard deviations away from the mean pitch. Thus, adding the contours simply means adding the standard deviations together.

A full description of the model is the code reproduced in 8.2. It uses MBROLA, which synthesizes voice given phonemes, timings, and a pitch contour.

The main data structure in the code is an utterance, which is composed of a list of utterances which should be concatenated and a pitch contour (which is a list of

[1] In fact, when on winter break, the author decided to experiment with this and take his sister's kazoo, an instrument through which one hums, and attempt to communicate by altering the pitch of his hum and modulating the volume with his hand. Simple ideas were not hard to convey.

pairs of times and pitches, where the times are percentages through the concatenated utterances).

The main pitch contours of the model are in `utter-oper`, which handles pitch contours of operators in a term-by-term manner, using general contours the volunteers used.

The current model is, as previously stated, very simple. A few improvements can be: adding "quantity" support to denote a quantity which has an operator of reduced precedence, being more natural than the pause doubling model; similarly, using "open parenthesis" and "close parethesis," which at least one volunteer used; prefix operators; summation notation; reordering of terms when there is an associative operator to simplify the utterance, which one volunteer did; and analyzing the complexities of various choices of utterances to find the clearest one.

It must be noted, though, that this ability to speak mathematical expressions clearly is not important to perfect since real people have a hard time doing this anyway, and the language can be very ambiguous. This model was developed to make it bearable to listen to the computer talk about such things.

It would be nice to integrate this model into an existing text-to-speech engine to give the system more words (the existing code only can speak ten different words, all hand coded) as well as enabling actual speech around the expression.

5.1.3 Speech Recognition

What has not been examined is the recognition of spoken mathematical expressions. It may be possible to interface with an existing speech recognition package and insert hooks for pitch to make the recognition more accurate. The pitch may actually help signal when a mathematical expression is happening.

6 An Interface

Ultimately, the goal of these recognition technologies is to be able to create a multimodal user interface for doing some kind of mathematical manipulation in a very natural manner. The interface will look like a large whiteboard. Equations can be written on the board. If the equations do not make sense, the computer will ask the user what they meant to write. Then, the user can ask the computer to help it simplify an expression, or to solve an equation for a particular variable using verbal statements such as "simplify this [pointing to the board] expression" or "solve this [pointing to the board] equation for x." The computer will then write the results on the board, perhaps commenting about the result, such as "here's an expression for x, but remember it must be greater than zero."

Also, to be natural, if the user verbally asks the computer to "factor $x^2 - x$," then the computer ought to also respond verbally with "it's $x(x - 1)$."

However, to fully understand the proper interaction, a study of mathematicians at a chalkboard should be carried out to better nail down the experience.

7 Future Work

Here are some ideas for (long-term) future work:

- **Geometry.** Using sketch recognition technology, geometric diagrams (as found in a geometry textbook or *The Elements* by Euclid) could be a main part of interaction. There is a grammar of symbols, denoting, for example, congruency.

  The system would have a knowledge of geometry, such as the fact that parallel lines never intersect (assuming Euclidean geometry, of course). When there are unfulfilled constraints in the diagram, the system could prod the user to fill in the gaps. Likewise, the user could ask the system to help find constraints.

  This system would entail developing a theorem prover system and encoding well known theorems from geometry.

- **Theorems and definitions.** In mathematics, words are defined in a specific way using previously defined words. Developing the interaction for telling the system something such as "an even number is a multiple of two" would be a great undertaking.

  A theorem is a statement about a property of an object if some conditions are satisfied. Similarly, it would be interesting to be able to describe a theorem to the computer. Even more interesting would be to be able to either explain or ask for an explanation of why a theorem is true.

- **Implement an undergraduate in mathematics.** With where the previous two items in the list were going, the author thought this was the most reasonable next step in the progression. Beware the person asking to get this kind of project funded.

8 Code

The following is code which was written to analyze data and test models.

8.1 avg-img.py

```
#!/usr/bin/python
from PIL import Image
import sys

infile = sys.argv[1]
outfile = sys.argv[2]

inp = Image.open(infile)
pixels = inp.load()

width, height = inp.size

colweights = [ sum([pixels[col, row][0]
                    for row in xrange(0, height)])
               for col in xrange(0, width)]

maxweight = max(colweights)/255.0

loc = 0.5

for col in xrange(0, width) :
    avg = 0.0
    moment = 0.0
    for row in xrange(0, height) :
        avg += pixels[col, row][0]/255.0
        moment += ((pixels[col, row][0] * row)/
                   (255.0 * height))
        pixels[col, row] = (0, 0, 0)
    if avg == 0.0 :
        pass
    else :
        # low pass filter on the pitch center
        loc = (avg/maxweight * (moment/avg)
            + (1-avg/maxweight) * loc)
        r = int(height*loc)
        v = int(255.0*avg/maxweight)
        pixels[col, r] = (v, v, v)

inp.save(outfile)
```

8.2 conv.lisp

```
;;; convert a mathematical expression
;;; into an mbrola file

(defvar *words* '())

; the phonemes for a word are a list of pairs
; of strings and timings.  Each string is an
; mbrola phoneme code
(defun add-word! (symb phonemes)
  (push (cons symb phonemes) *words*))
(defun get-word (word)
  (cdr (assoc word *words*)))

(add-word! 'a '(("EI" 160)))
(add-word! 'x '(("E" 100)
                ("k" 40)
                ("s" 60)))
(add-word! 'b '(("b" 40)
                ("i" 160)))
(add-word! '1 '(("w" 60)
                ("V" 90)
                ("n" 70)))
(add-word! '2 '(("t" 40)
                ("u" 210)))
(add-word! 'plus '(("p" 40)
                   ("l" 60)
                   ("V" 80)
                   ("s" 80)))
(add-word! 'times '(("t" 20)
                    ("AI" 70)
                    ("m" 70)
                    ("z" 80)))
(add-word! 'over '(("@U" 80)
                   ("v" 60)
                   ("r=" 100)))
(add-word! 'half '(("h" 40)
                   ("{" 80)
                   ("f" 80)))
(add-word! 'one-half '(("w" 40)
                       ("V" 75)
                       ("n" 50)
                       ("h" 40)
                       ("{" 80)
                       ("f" 80)))
(add-word! '_ '(("_" 120)))

(defun pause (time)
  (format '() "_ ~a~%" time))

;; a phoneme utterance is a list of
;; phoneme/timing pairs
(defun phoneme-utterance (phonemes)
  (list 'phoneme-utterance phonemes))
;; an utterance utterance is an utterance
;; which is composed of other utterances (for
;; instance, phoneme-utterances).  The pitches
;; argument is a list of pairs of times and
;; pitches.  The times are percentages through
;; the utterance.  The pitches are the standard
;; deviation from the mean.
(defun utterance-utterance (utterances pitches)
  (list 'utterance-utterance utterances pitches))

(defun rendered-utterance-length (rutt)
  (apply #'+ (mapcar #'cadr (car rutt))))

;; takes a list disps of (x_i y_i) pairs.
;; Linearly interpolates to find y for the
;; x argument
(defun lin-val (x disps)
  (let ((before (remove-if-not
                   #'(lambda (a)
                       (<= (car a) x))
                   disps))
        (after (remove-if-not
                  #'(lambda (a)
                      (>= (car a) x))
                  disps)))
    (cond ((null before)
           (cadar after))
          ((null after)
           (cadar (last before)))
          ((= x (caar after)) (cadar after))
          (t (let ((x1 (car (last before)))
```

```
                (x2 (car after)))
            (+ (cadr x1)
               (* (- x (car x1))
                  (/ (- (cadr x2)
                        (cadr x1))
                     (- (car x2)
                        (car x1))))))))))

;; the current value of the pause between
;; units of an expression
(defvar *pause* 1)
;; the minimum value of a pause seen (so pauses
;; can be normalized later)
(defvar *minimum-pause* 1)

(defun utter-relative-pause (a)
    (if (< a *minimum-pause*)
        (setf *minimum-pause* a))
    (list 'relative-pause a))

; given perc-displacements (a list of pairs of
; percentage/SD pitch displacement) and the
; length through which they act, add the linear
; interpolation to time-displacements (a list of
; time/SD displacement pairs)
(defun pitch-combine (len perc-displacements
                      time-displacements)
  (let ((disp (mapcar
               #'(lambda (d)
                   (list (/ (* (car d) len)
                            100)
                         (cadr d)))
               perc-displacements)))
    (let ((displaced
           (loop for td in time-displacements
              collect (list (car td)
                            (+ (cadr td)
                               (lin-val (car td)
                                        disp))))))
      (sort
       (append displaced
               (loop for td
                     in (remove-if
                         #'(lambda (a)
                             (assoc (car a)
                                    displaced))
                         disp)
                  collect
                  (list (car td)
                        (+ (cadr td)
                        (lin-val
                          (car td)
                          time-displacements)))))
       #'(lambda (a b)
           (<= (car a) (car b)))))))

;; takes an utterance, and outputs a data
;; structure which can be rendered by
;; render-phonemes
(defun render-utterance (utterance)
  (case (car utterance)

    ((relative-pause)
     (list (list
            (list "_"
                  (floor
                    (* 80
                       (/ (cadr utterance)
                          *minimum-pause*)))))
           '()))

    ((phoneme-utterance)
     (list (cadr utterance)
           (list
             '(0 0)
             (list (- (apply #'+
                             (mapcar #'cadr
                               (cadr utterance)))
                      1)
                   0))))

    ((utterance-utterance)
     (let* ((utterances (mapcar
                          #'render-utterance
                          (cadr utterance)))
            (lengths
              (mapcar
                #'rendered-utterance-length
                utterances)))
       (list (apply #'append
                    (mapcar #'car utterances))
             (pitch-combine
              (apply #'+ lengths)
              (caddr utterance)
              (apply
                #'append
                (loop
                  for u in utterances
                  for l in lengths
                  summing l into cum-len
                  collect
                  (mapcar #'(lambda (p)
                              (list (- (+ cum-len
                                          (car p))
                                       l)
                                    (cadr p)))
                          (cadr u)))))))))

;; takes data structure from render-utterance
;; and outputs mbrola codes
(defun render-phonemes (rend-utt mean sd)
  (setf *minimum-pause* *pause*)
  (format '() "~{~a~%~}"
          (loop for phon in (car rend-utt)
             summing (cadr phon) into loc
             collect
             (let* ((len (cadr phon))
                    (i (- loc len)))
               (format
                 '() "~a ~a~{~a~}"
                 (car phon)
                 (cadr phon)
                 (mapcar
                   #'(lambda (tim)
                       (format '()
                         " ~a ~a"
                         (floor
                           (* 100
                              (/ (- (car tim) i)
                                 len)))
```

```
                        (floor
                          (+ mean
                             (* sd
                                (cadr tim))))))
                    (remove-if-not
                      #'(lambda (tim)
                          (and (>= (car tim) i)
                               (<= (car tim) loc)))
                      (cadr rend-utt))))))))

;; takes a list of words and puts them together
;; as an utterance
(defun utter-words (&rest words)
  (utterance-utterance
    (mapcar
      #'(lambda (w)
          (phoneme-utterance (get-word w)))
      words)
    '((0 0))))

;; takes a mathematical expression and returns
;; an utterance representing the expression
(defun utter-math (expr)
  (let ((*pause* (/ *pause* 2)))
    (cond
      ((atom expr)
       (phoneme-utterance (get-word expr)))

      ((equal '(/ 1 2) expr)
       (phoneme-utterance (get-word 'one-half)))

      ((eq '+ (car expr))
       (utter-oper (cdr expr) 'plus))

      ((eq '* (car expr))
       (utter-oper (cdr expr) 'times))

      ((eq '/ (car expr))
       (utter-oper (cdr expr) 'over)))))

;; takes the arguments to an operator and the
;; operator itself and then returns an
;; utterance
(defun utter-oper (addends oper)
  (if (= 1 (length addends))
      (utterance-utterance
       (list (utter-math (car addends)))
       '((0 0) (99 -2)))

      (utterance-utterance
       (list
        (utterance-utterance
         (list (utter-math (car addends)))
         (if (= 2 (length addends))
             '((0 0) (30 0) (60 -1) (99 3))
             '((0 0) (50 1) (99 3))))
        (utterance-utterance
         (list (utter-relative-pause *pause*)
               (utter-words oper))
         '((0 2) (80 2) (99 4)))
        (utter-oper (cdr addends) oper))
       '((0 0) (99 0)))))

;; example test code
(let ((out (open "~conv.pho"
                 :direction :output
                 :if-exists :supersede)))
  (let ((str (render-phonemes
              (render-utterance
                (utter-math '(* (+ x 1) (+ a 2)))
                (utter-math '(+ (* 2 x) b))
               (utter-math '(+ (/ 1 2) x))
               )
              240 20)))
    (format t "~a" str)
    (format out "~a" str)
    (format out "~a" (pause 120)))
  (close out))

(sb-ext:run-program
  "/home/kmill/mbrola/mbrola-linux-i386"
  '("/home/kmill/mbrola/us1/us1"
    "~conv.pho"
    "~conv.wav"))
(sb-ext:run-program
  "/usr/bin/mplayer"
  '("~conv.wav"))
```

9 References

D. Blostein and A. Grbavec. *Recognition of Mathematical Notation*. Handbook on Optical Character Recognition and Document Image Analysis.

E. Tapia and Raúl Rojas. *Recognition of On-Line Handwritten Mathematical Systems in the E-Chalk System*. ICDAR 2003.

J. LaViola Jr. and R. Jeleznik. *MathPad^2: A System for the Creation and Exploration of Mathematical Sketches*. ACM.

K. Chan and D. Yeung. *Mathematical expression recognition: a survey*. IJDAR 2000.

N.E. Matsakis. *Recognition of Handwritten Mathematical Expressions*. Thesis, Massachusettes Institute of Technology. 1999.

R. Zanibbi. *Recognition af Mathematics Notation via Computer Using Baseline Structure*. 2000.