How can we speak math?

Richard Fateman
Computer Science Division, EECS Department
University of California at Berkeley

June 18, 2004

Abstract

Surprisingly speech is a good adjunct to other ways for a human to enter mathematical formulas into a computer. Speaking, especially in small pieces, provides an independent channel or "mode" that can be combined with error-prone modes including pointing or handwriting. This combination may allow us to identify and cancel errors of handwriting and result in more reliable communication. It may also be more convenient than typing, even for rapid typists, since many mathematical symbols are easily spoken but are not on the keyboard. Some classes of these tokens cannot be easily distinguished from each other merely by handwriting.

Pursuing this goal of effectively speaking small pieces of mathematics, we wondered how hard it would be to speak rather longer sections of mathematics, including nested complex expressions.

We first describe programs for the computer-generation of mathematical speech, Suggesting a few speaking conventions that overcome the unfortunately ambiguous and inconsistent common usages.

Then we consider tools and guidelines to make it more plausible for humans to speak mathematical formulas so they can be recognized by a computer speech recognizer. We describe prototype programs.

1 Introduction

Handwriting mathematics seems natural because it is what we have been taught. For the same reason we find it natural to view mathematics in typeset form. Speaking mathematics seems unnatural, but in fact users of math routinely speak small pieces quite comfortably. Given that speech input to computers is becoming more common as it is better supported by technical advances, the question arises: can we speak mathematics into the computer? If we could do so, persons with disabilities in writing or typing, should be able to transmit mathematics. Even for non-disabled, there may be advantages for speech in some circumstances; we contend that speech can be used as a primary method, a supportive auxiliary method, or an error-correction command language.

The reverse: a computer speaking mathematics, has more of a history, and some notable successes such as AsTeR [13] and Design Sciences' MathPlayer.

We first look at this material in order to see how a mechanism could speak mathematics. We then proceed to our main results where humans speak aloud and the computer listens to mathematical discourse.

2 Computers speaking math

The program AsTeR [13] is an excellent prototype for speaking mathematics; indeed it seems quite worthy of use for the reading of TEX mathematics to visually disabled persons. There is a problem with TEX that cannot be easily overcome in that there is no way for AsTeR to be given the semantics for the material from TEX except from some (external) context. Thus f^{-1} might be f to the power -1 or it might be f inverse, or even, in the case $\sin^{-1}$, the function named arcsine. If the speech is generated from a computer algebra system, or encoded in a semantic description in MathML, there is a better chance of getting it right. In fact, Design Science, www.dessci.com has a "speak expression" option that allows Internet Explorer to read math aloud from a MathML expression if the (free) MathPlayer plug-in is available. Its effectiveness seems to vary depending on the computer setup. It seems to use locutions like " begin fraction a+b over c+d end fraction." It seems to us plausible to have an advantage if one directly speaks from a computer algebra system rather than through a browser. In the CAS case, the system could contain more context including line labeling schemes, aliasing of symbols to names, or abbreviations (e.g. let $r = \sqrt(x^2 + y^2)$ in an expression). It could also make reasonable and consistent choices as for x^{-1} vs. $1/x$. It might even describe expressions to prepare the listener. For example "a fraction with a long numerator of 25 summands and a denominator the product of 5 terms, namely: ...."

2.1 Speaking on the Internet

Given the state of the art today, it is possible to have a web browser speak (in one of the various available voice of your chosing) the xml encoding of a speech utterance. We have experimented with this, and have written a program suite providing the translation of algebraic expressions as Lisp prefix data into words. For example `(* r s t)` would be spoken as "r times s times t." More specifically our lisp-to-speech XML program would produce this code underlying code for $r \cdot s \cdot t$.
`<p><spell>r</spell> times <spell>s</spell> times <spell>t</spell> </p>"`
Similarly, $f(x, y)$ would be `"<p><spell>f</spell>  of <spell>x</spell> and <spell>y</spell> </p`

Not all the nuances of AsTeR may be available, but the encoding provides for changes in volume, speed and pitch. An additional feature using stereo might be neat, proceeding from the left speaker to the right as the expression is read aloud.

A curiosity that we did not anticipate in our initial design is the extent to which most listeners and speakers leave out critical information, even when they think they are speaking unambiguously, and how overbearing the complete and unambiguous rendering sounds.

The well-known quadratic formula can be written as a Lisp prefix expression as `(/ (pm (- b) (^(- (b^ 2) (* 4 a c)) 1/2)) (* 2 a))` where `pm` means $\pm$. This can be read in a variety of ways. Here we remove the `<spell>` pieces, as well as a change in pitch for the denominator and other minor items, in order to make the text more perspicuous: `minus b plus or minus square root of b squared minus 4 times a times c divided by 2 times a .`

But how do you know if the $2a$ belongs within the square root? You don't from this reading. Is the $-b$ in the numerator or outside the fraction? Again you don't know. In fact, is the a in the denominator, or is it a multiplier for the whole previous expression?

The overbearing program that insists on bracketing, by inserting "the quantity" and "end" around components can provide non-ambiguous renderings. But for this formula it needs to put in three sets of brackets, making it seem excessively pedantic. Judicious omission of bracketing on output seems advantageous, and so our program does not always insert brackets. Instead there is a explicit insertion of tags required for enuniating brackets. As an example (* (+ a b) c) could be confused with (+ a (* b c)), and so one first should use a bracket constructor to be spoken by the computer (and to keep the listener on guard). The example would be (* (bracket (+ a b)) c), and would be pronounced "The quantity a plus b end times c."

2.2 Reading numbers aloud

At the base of mathematics, we need to be able to read numbers. It came as something of a surprise that the reading (and hence the speaking) of numbers is so rife with special cases and ambiguity.

The TTS program from Microsoft which we use has some interesting features for reading numbers aloud. We review its behavior not only for amusement, but for education. After all, if we hope to have the computer listen to us speak numbers, perhaps we should attempt to understand the rules that TTS uses for pronouncing numbers (starting from text) as guidelines.

The following examples (from speech SDK 5.1) suggest that sometimes this provides a plausible guideline, but we have no access look at the complete ruleset for TTS. How does TTS speak numbers given to it as ascii text?

Here are some examples.

- 123 is one hundred twenty-three.
- 123.123 is one hundred twenty-three point one two three
- 1,000.00 is one thousand.(*)
- 1,000.000 is one thousand point zero zero zero.
- 3.1415929 is three point one four one five nine two six
- 3.14.15929 is three point fourteen point fifteen thousand nine hundred twenty-six. (*)

- 3.14.1592 is March fourteenth, fifteen ninety-two. (Note the use of ordinal 14th).(*) The program knows that the nearby "number" 3.32.1592 is an invalid date, and thus spells it out. It does not actually know that September has only 30 days, much less the rules about leap years. In fact it is not possible to speak this into the standard dictation grammar, which will produce a sequence of two numbers, 3.14 and 0.1592. But see the related date *fractions* below.
- 1/10 is one tenth.
- 9/10 is nine tenths.
- 10/11 is ten over eleven.
- 14/100 is fourteen hundredths.
- 14/10000 is fourteen over ten thousand.
- 14/100000 is fourteen slash ten oh oh oh oh. (*)
- 14/1000000 is fourteen slash one oh oh oh oh oh oh. (*)
- 14/100000000000000 is fourteen slash one zero zero ... zero.
- 14/ 100000000000000 is fourteen slash ten trillion.
- 3/100 and 300 sound almost the same: "three hundredths" versus "three hundred."
- 2-2 as well as 2-2-2 is two to/two two.
- 1-3, as well as 1-2-3, is one to/two three.
- 1-2-9 is one two nine, but 1-2-10 is January second, ten.
- 40/500 and 45/100 are indistinguishable. (The second can only be spoken as 45 slash 100 or 45 over 100. forty-five hundredths yields 40/500.)
- 3/14/1592 which might appear to be (3/14) divided by 1592, is not. It is March 14, 1592.
- 0.0 is zero point zero.
- 0.00 is just zero.
- 1,500,000 is 1 point 5 million.

Integers up to "999999999999999" (999 trillion and change) are spoken, but above that are spelled out digit by digit. There are different rules for integers appearing in denominators.

Numbers that do not have commas set out correctly are spelled out. Thus 5,10.0 is five comma ten point zero.

Floating point numbers such as "5.00d0" are handled as separate components, namely "5.00" or five, and "d0" (dee zero). -1/2 is dash one slash two.

2.3 Non-speech approaches to natural math

This is necessarily a brief review. On the *output side*, in recent years computers have increasingly been applied to the task of mathematical typesetting, now supporting the whole workflow from the original creation by an author/ typist, through interpretation by some typesetting program, to the point of printing on paper or display to a reader. Most readers of this paper will be aware of such editors (using keyboard and mouse) and printers (using raster graphics).

On the *input side*, since at least 1965 programs have been demonstrated which serve as intermediaries for the conversion of (hand)written material into typeset material. More recently it has become plausible to actually make use of such programs on the much more powerful computers of today.

Today's demonstration programs [15, 11, 2] show that while it is fairly easy to recognize a subset of simple math symbols and expressions as usually written by hand, there are substantial problems. A high error rate on simple tasks is a consequence of understandable difficulties. Trouble distinguishing many pairs: (p vs P, 0 vs O, 5 vs S etc), means that some demonstration programs take drastic approaches such as simply excluding the letters S,l, and O from the vocabulary. Other confusions are possible with positioning or stroke identification. Thus `1<2` might be confused with `K 2`.

We have an experiment that might illustrate some of the difficulties, easily performed by a college student or teacher.

> Walk in to a mathematics or physics classroom at the end of the lecture and see if you can read all the mathematics on the blackboard.

You probably can't understand it all. Expecting a computer to understand it, devoid of mathematical or physical context, is unrealistic. Additionally, a computer post-processing of a blackboard has another handicap compared to the student in the classroom. The computer does not have the benefit of the lecturer's simultaneous speech while writing on the board, the sequence of writings and erasures, and of course does not have the opportunity to ask clarifying questions.

Other methods for input and editing of math via templates, menus, keyboarding, and other non-handwriting forms are surveyed by Kajler and Soiffer [7]. A skilled user can generally do quite well with such systems, but systems can be frustrating to the novice. In some cases they can also be frustrating to the expert who requires close adherence to some format unanticipated by the system designers. Our proposals generally would complement such existing systems, especially TEX and TeXmacs, an interactive system inspired by TEX.

One reviewer of an earlier version of this paper claimed that "the use of dual input (speech and pen) is not much different than pen and keyboard or pen and palets[sic]", missing the point that one cannot type on a keyboard while using a pen or a mouse. But one can use a pen and speak.

2.4 Speaking mathematics

We are studying how to provide mathematics communication input. This could be used for a spectrum of uses from education to preparation and search in digital systems, as well as interaction with web-based services. In our case we are looking at "multimodal" techniques of speech in combination with handwriting [5, 4]. As a preliminary question concerning aids to written mathematics, it seems prudent to address an apparently simple question: How do we speak mathematics?

2.4.1 Brief digression on speech processing

There are substantial research efforts on speech-related topics. From a relatively naive standpoint, but one which we think is adequate here, the speech issues seem to be separable into

- Output aids for the visually impaired. The audience may be computer users (programmers, too) who type using a keyboard but are unable to see text as routinely displayed by a computer. Text-to-Speech (TTS) makes it possible for a computer to "read aloud" to a blind person, or to speak to a person who has no other display. The primary target here is a person using a telephone. A truly useful audio interface for a structured domain like mathematics or a graphical display will require rather more elaborate design [13] than just reading a text.

- Input aids to the keyboard-typing impaired. The user may suffer from some temporary or permanent disability. Automatic Speech Recognition (ASR) makes it possible for a user to "speak" words and phrases, constituting dictation of content or commands to the computer. Generally the user is able to see a display, but not always. A user of such a system might be at a telephone, speaking commands to a computer. (Simple numeric input might best be provided through the telephone keypad, at least for a sighted person. Alphabetic input is trickier.)

- "Multimodal" assistance, for example for the task of correction (proofreading) of material that may have been entererd into the computer by some error-prone method. The first method might be document image analysis, handwriting, or speech. Both TTS and ASR may be used. Proofreading data entry, for example of tables of numbers, by having them read back by the computer seems quite straightforward with today's technology. Even reading math formulas out loud to see if they have been typed (or typeset) could be an application.

There are also distinctions in whether one has a system trained on a single voice or must work with all speakers (harder), and whether the vocabulary and grammar is simple or large.

2.4.2 The trivial non-solutions

One solution for "speaking mathematics" that immediately presents itself as unambiguous is to merely spell expressions as though you were typing them character by character on a single line. All the disambiguation must be done prior to spelling. In this way the problem has been reduced to that of the previously "solved" problem, namely the parsing of a programming language that is typed into a computer, and all that is needed is a mapping of sound to keyboard element. If the encoding language is TEX, then the appearance of almost any mathematical notation can be provided, on almost any computer system, thanks to the continuing work on maintaining TEX. If the programming language is the painfully verbose semantic MathML, simulating a keyboard by voice would be very time-consuming, but even with TEX, entering β would require saying something like "dollar backslash b e t a dollar" or once you realize how close certain sounds are (a, eight) or (b, d, p) or (s, f), you might use a "military alphabet" for spelling. (In practice a military[1] spelling option is slightly wordier but nearly error-free.) Thus for a higher accuracy, you might learn to say "dollar backslash bravo echo tango able dollar". Compare that to saying "beta"!

(We note that the usual programming language notations, such as Fortran, are grossly inadequate notationally for serious math, and we cannot seriously consider "speaking Fortran" as a substitute for math[2]. We also note that the interpretation of TEX as math can be ambiguous, but at least nit is as good as mathematicians usually see; a spoken version will not necessarily be semantically unambiguous either!)

3 Developing an intuitive speech model

3.1 How humans should speak numbers

This seems to be trivial, but has complications if you want to use numbers like "three and a quarter" or "one point five million." Our advice is to use easily-parsed "full" natural numbers including properly indicated steps like "one hundred twenty three thousand". An alternative is a string of single digits. Full numbers may be combined with decimal points ("." pronounced "point") or for fractions, the virgule ("/" pronounced "slash" or "over"). We also permit "oh" for zero. The main difficulty we see in the purely digit-list prescription is that some numbers, like 3 million, have an excessive number of zeros to pronounce and recognize accurately.

There are other problems if numbers occur adjacent without intervening punctuation. This can happen with single digits perhaps more often: "The

[1] NATO uses Alpha Bravo Charlie Delta Echo Foxtrot Golf Hotel India Juliet Kilo Lima Mike November Oscar Papa Quebec Romeo Sierra Tango Uniform Victor Whiskey Xray Yankee Zulu.

[2] Of course, speaking Fortran qua Fortran, or using speech as source input in any programming language is a possibility, with many difficulties not necessarily related to math.

single-digit primes are 2, 3, 5, and 7" does not mean "The single-digit primes are 235 and 7." Thus the commas must be enunciated, or the speaker must force the recognizer to accept the phrase in pieces. "US paper currency includes fifty, one-hundred and five-hundred dollar denominations" could be read as "5100 and 500 dollar."

We tried several approaches.

- A pattern-matching heuristic program we have written is perfectly happy with numbers constructed like "one hundred twenty-three thousand four hundred fifty-six point seven eight" for 123,456.78. We recommend "one slash two" for 1/2, since generalizations of fractions are tricky. Being written in Common Lisp, it has no limits on the number of digits possible.

- For most uses, we expect that the Microsoft published `cmnrules` grammar[3] for various kinds of numbers including natural numbers, fractions, floating-point, could be used. Much to our relief this can be included rather painlessly in a speech recognition program by specifying (in an SASDK/ SALT application that can, for example, be run with a browswer plug-in), a `listen` tag. `<item> <ruleref uri="cmnrules.grxml#number"/><tag> $._value = $$._value`

- The principal defect in `cmnrules` from our exact mathematics perspective is that it is limited to numbers less than (10^{15}) and fractions are converted to decimal numbers of limited precision. This is an artifact of using the arithmetic in the underlying J++ scripting language which is the default (and at the moment sole) programming technology in the Microsoft grammar implementation of the W3C recommendations for XML speech grammar. We have constructed a modification of the grammar to maintain exact ratios for numbers like 1/3, where numerator and denominator can only be represented exactly by strings. This is passed on to lisp for further evaluation. Thus the string "six quintillion plus one" is parsed to "(+ (* 6 (expt 10 18)) 1)" which is exactly evaluable in Lisp. (There is a disappointment at a different level in the grammar XML processing, for which, see the section ?below.

- A third possibility, also easily implemented by reference to `cmnrules` is to use lists of digits for numbers. This is occasionally in conflict with the other common usage rules, but could easily be used instead of, or in preference to the more general usage. In fact the digit-list convention is used in conjunction with other parts of the grammar for decimal fractions. Consider "seventeen hundred point oh four five". To the right of the point we speak in digit lists.

3.2 Non-numeric tokens

In our experiments to date it seems important to realize that a large word list, in which context does not provide strong clues, that Given the presence of

[3]We found, reported and corrected two bugs in this. June, 2004.

easily-confused phonemes, we have a choice.

- Satisfaction with relative poor initial accuracy, relying on rapid correction.
- Resolution of ambiguity based on context. Given our formula context, we prefer "eight equals two times four" to the identical phonemes in "ate equals to times for". Unfortunately "Pick a number from one to ten" and "Pick a number from 1, 2, 10." are rather close. Sometimes it is easier. "Capital a" is plausible. "Capital 8" is not.
- Removing some ambiguity at the source: to rename some or optionally all letters via a "military alphabet". We choose names one that do not conflict with other math tokens such as Greek letters. Thus (adam, ..., david, ...) rather than (alpha, ..., delta, ...).

Other token considerations:

The well-used spoken tokens include not only letters of the Roman alphabet (perhaps modified with "bold," "roman," "italic," etc), but other alphabets as well. Symbols taken from sources include the TEX typesetting repertoire, computer algebra systems such as Mathematica, and Unicode. Even among the common names, there are ambiguities. Consider the tokens for "sign" and "sin" which are both plausible in many contexts.

Words for spaces are handy as well, such as "quadspace".

Typically these tokens can be separated into operators and operands, but we cannot depend on such classifications for rigid parsing.

It is also quite likely that macro-expressions defined verbally will be useful for the serious speaker. Thus " let foobar equal script capital bold a sub Greek nu" allows foobar as an abbreviation. Clearly this could be made as elaborate as any macro language, although here we propose simple constant non-parametric substitutions.

3.3 Caution on complete forms

Imagine how annoying it would be if, as you were typing at a computer keyboard, every one of your pauses were treated as an end-of-sentence marker, and the computer immediately made an observation that your sentence was incomplete, or if it appears to be complete, the computer whisked it off to process. We must refrain from insisting that math be spoken all in one breath. We can signal explicitly by a mouse click[4] or alternatively, the computer will just wait, and proceed after a short pause when you are presumed to be finished speaking *for the moment*. In such circumstances it cannot be too authoritarian about preventing what you say next to be added to, or somehow modify the previous utterance[5]

[4]We can signal the end of a phrase by a word marker such as "OK", but the recognizer will not necessarily process it completely unless there is a pause, also.

[5](What's your favorite color? Blue. No, yellow; http://www.sacred-texts.com/neu/mphg/mphg.htm)

3.4 Expressions

In this section we describe variations for speaking a prototypical expression that would seem to be at first glance non-linear in appearance:

$$\frac{a+b}{c+d}$$

which however be linearized in various ways. In TEX it is spelled out as `$\frac{a+b}{c+d}$`...

Or spelling it out we could say, "dollar, backslash eff arr ay see open brace, ay plus bee close ...". In a military alphabet ... foxtrot romeo adam charlie ....

We assume here that "close" is adequate to match the previous still-open bracket, and we can save quite a few syllables if we do not have to say "close parenthesis" or "right parenthesis".

In future examples in this paper we won't use spelling, even though it may be inevitable for peculiar words.

Instead of spelling TEX we can spell a linearized form (a+b)/(c+d), which is shorter, unambigous, but still uncomfortable. Instead of a dollar sign we use "begin math" and "end math". Instead of targeting TEX we are targeting a typical programming language (perhaps a computer algebra system.)

```
begin math ( a + b ) / ( c + d ) end math.
```

This requires saying open/close four times. To preview our proposal, in this paper we suggest that the expression above be spoken this way:

```
begin math
a+b quantity
over
quantity  c+d
end math.
```

or perhaps

```
begin math
adam + bravo quantity
over
quantity  charlie + david
end math.
```

(We will refrain from using the military alphabet subsequently because it is a distraction; however, failures in recognition of simple letters can be easily remedied this way.)

Grouping based on the embedded key words *quantity*, *over* and *end* can be done by some simple transformations on the stream of tokens We start by implicitly enclosing every begin/end math expression with a default $(\cdots($ and $)\cdots)$. The word "quantity" immediately *after* an *operator* (defined below), can be changed to the insertion of a "(". "Quantity" *before* an operator, is

equivalent to ")". If the speaker says "quantity" between two operands (which are presumably going to be multiplied together by a "silent times") then we propose the same result as "quantity times quantity". This may not be the speaker's intention, so some extra feedback or warning may be advisable.

The extra prefix "(" and suffix ")" are appended only as needed to balance the brackets. Operators include

- infix such as plus, times, over, slash, divided-by, raised-to, space, quadspace
- prefix such as sum, product, function of (e.g. sine of),
- suffix such as factorial, squared, cubed, prime, double prime,
- overhead, which in TEX constitute prefix such as hat, bar. In common math speech, these would generally be voiced as suffix operations. $\hat{x}$ in TeX is `$\hat{x}$` but probably pronounced `x hat`.
- matchfix such as left/right square brackets, left/right angle brackets. These can come in many sizes like `big` or `big big`.

There are large tables of additional operators in The TEXbook, and similar references attempting to be encyclopedic, including Mathematica.

Typical operands are essentially everything else, including syntactic components like symbols, numbers, and (recursively) subexpressions.

Given these rules, our spoken expression is transformed to text as

```
(a+b)
/
(c+d)
```

3.5 Math on a line

It might seem that any math expression that fits on a single line without up/down excursions would not be problematical, since it has an "obvious" order in which to read characters[6]. It seems that difficulties could only occur if the speaker leaves out characters necessary for grouping, or declines to pronounce the brackets. Unfortunately, leaving out such characters is entirely conventional, as shown by later examples.

Simple Examples:

Display	Spoken
$ab \sin x$	a b sine of x
$a + \frac{b}{c} + d$	a + b over c + d
$\frac{a+b}{c} + d$	a + b quantity over c + d
$a + \frac{b}{c+d}$	a + b over quantity c + d

[6] Actually a linear sequence is possibly ambiguous in a larger sense of conveying mathematics. $1/2\pi$ sometimes means $\pi/2$ and sometimes $1/(2 \times \pi)$. But this is not a speech problem.

This next set of examples is insufficient to tell us how to deal with extra cases that require groupings "in the middle".

Most of what we have said up to this point does not get much of a rise out of most readers. This next proposal is more controversial: We believe we have to add only *one* additional linguistic marker, *all*, or alternatively, *end* or *close*. Let us argue in favor of this.

The term "all" or its alternatives essentially *jumps out a level.*

Display	Spoken
$a + \frac{b}{c+d} + e$	a + b over quantity c + d all + e
$a + \frac{b}{c+d} + e$	a + b over quantity c + d all times e

We can also use "all" without "quantity"

Display	Spoken
$(a + b)/c + d$	a + b all over quantity c + d

Consider this: $a + \frac{b}{c+d} \times \frac{e}{f} + g$. We could try grouping this by pauses: a + pause b over quantity c + d pause times e over f pause + g. Raman's AsTeR program [13] can use speech cues for output, but speakers, and the programs listening to them may not be so disciplined. And sometimes one would need several pauses at the same place. Nevertheless, in combination with a geometric handwriting interface and feedback, perhaps this could work.

Display	Spoken
$a + \frac{b}{c+d} \times \frac{e}{f} + g$	a + quantity b over quantity c + d all times e over f + g
$(a + \frac{b}{c+d}) \times \frac{e}{f} + g$	a + b over quantity c + d all all times e over f + g

This last expression is peculiar in requiring "all all", but we see no especially intuitive shorthand around this occasional need. No one said that reading mathematics, especially deeply-nested mathematics, was going to be simple!

Let us return to the quadratic formula. We can say it "The quantity minus b plus or minus the square-root of the quantity b squared minus 4 a c end all over the quantity 2 a end."

4 Speaking Integrals and Sums

The integral [from x=a to b] of f(x)+g(x) d x has the advantage of the closing "d x", and so in most (not all) traditional notations we can try to read or listen ahead looking for the "d".

The $\sum f(i)$ construction doesn't have any close, so $\sum fg + h$ is ambiguous. We could just leave it that way and say that our job is over when the speech is changed to text, but can we fix it with a modest effort? It is unlikely to mean
$(\sum f) \times g + h$
but could be $(\sum f \times g) + h$

or $\sum(f \times g + h)$.
These could be spoken, respectively as
sum of f all times g + h;
sum of f times g all + h;
sum of f times g+h all.
Of these, the last seems a strain, but only because there is no operator after the h. If this is truly the end of the expression, the "all" could be left out! The others seem fairly natural, and perhaps more natural if we allow "f times g" to be simply "f g".

The advantage of a multimodal input model is that the computer system can display what has been recognized. The longest delay is likely to be the pause while the computer waits to determine the end of the utterance. The translation and typesetting should be quite rapid by comparison. Thus the feedback of the choice made by the system may provide a valuable learning experience in these less common forms.

5 Additional Examples

We promised some additional examples to fill out the description.

Display	Spoken
$a_0 + x(a_1 + x(a_2 + \cdots))$	a sub 0 + x times quantity a sub 1 + x times quantity a sub 2 + dot dot dot
$((a_3x + a_2)x + a_1)x + a_0$	a sub 3 x + a sub 2 quantity times x + a sub 1 quantity times x s+ a sub 0
$a_{(n-1)^2} + 1$	a sub quantity n minus 1 all squared all + 1
$a_n^2 - a_{n-1}^2$	"a sub n squared minus quantity a sub quantity n minus 1 all all squared
$a_n - a_{n-1}^2$	quantity a sub n minus a sub quantity n minus 1 all squared
$(a_n - a_{n-1})^2$	a sub n minus a sub quantity n minus 1 all all squared
x^3	x to the third (note ordinal 3rd)
	x to the third power
	x to the power 3
	x raised to the power 3
	x cubed

For convenience in the next few examples we will just say "x ↑ 3" for x^3.

Display	Spoken
$x^{34}e^z$	x ↑ 34 e ↑ z
	x ↑ 34 times e ↑ z
$x^{n+1}e^z$	x ↑ quantity n + 1 all times e ↑ z"
$x^{1+n^{2+k}+r}e^z$	x ↑ quantity 1 + n ↑ quantity 2 + k all + r all times e ↑ z
$\frac{x^n}{y^m} + 4$	x ↑ n over y ↑ m + 4
$f(x) + g(y, z) + r_{i,j} + h(s_i, j)$	f of x + g of y comma z all + r sub i comma j + h of quantity s sub i all comma j

The next section presents a small controversy as to how to bracket argument lists of functions. Consider either form $\sin x$ or $\sin(x)$ can be spoken "sine of x" or "sine x". The comparable "anonymous" functional application form, assuming there is a function named p could be written px or $p(x)$. Either of these can be "p of x", but only the first of these, px can plausibly be spoken as "p x". And in that case it might be the product of two items. What gives?

Here is one proposal:

There is *no special meaning for "of" in* "sine of ... " or any function known to be a univalent function. If the single argument is compound it must introduced by "open" or "quantity" Thus $\sin x$ is simply "sine x" but $\sin(2x)$ is "sine of quantity 2 x [close]".

If the function is not well known, then there is a significance to the "of". px should be pronounced "p of x" and probably should be written $p(x)$. Saying "p x" is hazardous. It looks like a multiplication. This is not a prohibition; spoken math as well as written math can be ambiguous! "p of x + 1" means $p(x) + 1$. "p of quantity x + 1 close + 2" means $p(x + 1) + 2$.

Display	Spoken
$a \sin x$	a sine of x
	a sine x
$a \sin x + y$	a sine of x + y
	a sine x + y
$\frac{a \sin(x+y)}{2} + 1$	a sine of quantity x + y all quantity over 2+1
$a \sin \frac{x+y}{2} + 1$	a sine of quantity x + y quantity over 2 all + 1
$\sin x + \cos y$	sine of x + cos of y
$\sin x + \cos y$	sine x + cos y
$\sin(x \cos y)$	sine quantity x + cos y [unusual]
$f(x, y) + 1$	f of quantity x comma y all + 1 [bad]

versus Proposal 2: The word "of" has special significance and always carries with it an implicit "open". Thus "p of x + 1" means $p(x + 1)$, and "p of x + 1 close + 2" means $p(x + 1) + 2$.

The alternative here allows a distinction between "sine of x" and "sin x", but that may be too subtle for users/ speakers.

Display	Spoken
$a \sin x$	a sine of x [implicit close]
	a sine x
$a \sin x + y$	a sine of x all + y
$\frac{a \sin(x+y)}{2} + 1$	a sine of x + y all quantity over 2 all + 1
$a \sin \frac{x+y}{2} + 1$	a sine of x + y quantity over 2 close + 1
$\sin x + \cos y$	sine of x all + cos of y all
$\sin x + \cos y$	sine x + cos y
$\sin(x \cos y)$	sine of x + cos y close [unusual]
$f(x, y) + 1$	f of x comma y all + 1 [better]

6 Extensions

You may not be entirely comfortable with the limited vocabulary, and prefer other words and phrases. These should also be allowed, certainly to the extent that they do not interfere with the existing mechanisms.

Other notations such as "the fraction a + b divided by c" are easily accomodated. The word "fraction" has exactly the same meaning as "quantity", and the phrase "divided by" means the same as "over".

We might plausibly say, and parse, $(a + bc)(d + ef) + 1$ as "the product of a + b times c and d + e times f all +1" as an alternative to "a + b c quantity times quantity d + e f all +1."

There are other common locutions such as "d squared by d x squared of f of x" for $\frac{d^2}{dx^2} f(x)$ which could be spoken as "d squared divided by quantity d x squared all f of x" or $\frac{d^2 f(x)}{dx^2}$ which could be spoken as " d squared f of x quantity divided by quantity d x squared." A phrase such as "the second derivative of f of x with respect to x" could be worked into a parser.

Here are some example additional phrases. Consider $y'' + a^2 y = 0$ spoken as " y double prime + a squared y equals 0". Thus "prime", "double prime", and "triple prime" seem like "prime candidate" for additions.

To show the flexibility or perversity of notation, here is an expression approximating original notation from Hoare logic [6] describing the semantics of **while** loops. It looks like this:

$$\frac{P \wedge b \ \{S\} \ P}{P \ \{\textbf{while } b \textbf{ do } S\} P \wedge \neg b}$$

Note that adjacent symbols here are not multiplied but in the form $A\{B\}C$ indicates a kind of temporal ordering, proceeding from left to right. The curly braces have specific meanings (separating predicates from program sections), and the horizontal line means something like "implies" and does not have any relationship with division. We can nevertheless speak it as " cap p wedge b { cap s } cap p quantity over quantity cap p { roman while b roman do cap s} cap P wedge neg b".

Any programming structure for the recognition of math must be extensible in two directions:

- Allowing new spoken tokens to be introduced to the speech grammar, and
- Allowing new phrases to be parsed, at least in a rudimentary fashion.

Thus one might need to add the words "Poisson bracket" and appropriate pronunciation to the speech engine and then also introduce a grammar rule to allow the "Poisson bracket of x and y" to be typeset in its usual notation (x, y), or perhaps come up with some (almost any!) more perspicuous form.

The spoken form as we have specified it does not have any precedence rules of its own, and perhaps surprisingly tends to just pass along ambiguities: That is, you can often speak or typeset an expression without complete concern for its meaning. For example, $a = b \vee c$ could denote one of the Boolean expressions $(a = b) \vee c$, or $a = (b \vee c)$, or could be an assignment expression, or something else. Even as we use the expressions to convey different meanings to the reader in the previous sentence, we feel compelled to point out that it requires some kind of external interpretion to distinguish those expressions. When computer typesetting was more of a novelty, it provided the unfortunate illusion that a formula undergoing typesetting gains authority; as is obvious to the modern reader, typesetting does not mean correct or even defined.

6.1 A disappointment

It would appear that the speech grammar XML formalism provided by the W3C organization and its conforming implementation by Microsoft would naturally provide a framework for context-free grammar parsing. In fact, Microsoft uses a file-extension ".cfg" for compiled grammars. Sadly, the grammars are not permitted to be context-free, but only finite-state. This means that a grammar that allows for nested subexpressions, (using "quantity" and "end" or equivalently open and close parentheses) *cannot* be described in the given speech grammar. This throws us back to a more primitive stage in which we can use the microsoft grammar for recognizing tokens (numbers, symbols), but cannot actually depend on it for parsing. It is true that with some effort one can write a grammar that looks like it will work for expressions, but only if they are of finite depth. Thus one could come up with a grammar that allowed no parentheses, or some fixed depth. This requires essentially duplicating the grammar for each nested depth. W3C *permits* but does not require a context free grammar. We do not know if the Microsoft implementation will exceed its current capabilities.

7 Comparisons

A commercial computer program, Mathtalk [9] is an attempt to provide a facility for humans to speak math to a computer. The on-line demonstrations suggest that the method used is rather sluggish, requiring pauses before and after each

operation to wait for recognition. It is not clear how to correct any mistakes. The engineering is quite limiting: Mathtalk *requires* referring to letters by military names (e.g. f becomes "function foxtrot" and $\lim_{n->\infty}$ is "limit November goes to infinity".) It is unclear how much of the limitations are inherent in the design or in the lower-level support from their perhaps primitive speech tool.

8 Status of spoken math recognition

We have written a rudimentary translator from strings of some spoken symbols into strings again, but of more conventional symbols, essentially replacing words like "quantity" and "all" by parentheses, and inserting some parentheses in other places as appropriate. We can also parse numbers from spoken words (twenty one hundred becomes 2100). Since we anticipate that words and symbols will be misrecognized or missed entirely, we cannot just walk away from the task when the speaker halts. There is a feedback step in which the computer attempts to display—to the extent possible—what has been recognized or not. This feedback is described in a separate papers on the display of incomplete expressions [4]. This feedback is based on transforming a string of tokens into TEX and typesetting an approximation to what has been spoken, with placeholders for parts that were not recognized. It may seem odd to a programming-language trained reader that one can truthfully declare complete speech recognition success upon nicely typesetting something "symbolic" as a mere string of words. Yet it cannot be the recognizer's fault if the human speaker has uttered partial or complete nonsense posing as mathematics. Given some partial display, it may be plausible for the speaker to abandon the original utterance and instead patch what the computer heard. If an unmatched open parenthesis is spoken, the matching one will be displayed, and need not be spoken. However, the insides may need to be filled in.

As of June, 2004, K. Lin and R. Fateman connected a speech recognizer with a simple grammar to a prototype mathematics editing system (SKEME), which allows the user to speak simple tokens or expressions instead of typing. The tokens may be compounded as in "script capital A" and they may also be concatenated with simple operators as in "a plus b". The locations for the insertion of spoken or written symbols is governed by cursor or attention point which is positioned with a mouse. The mechanism used (Microsoft Speech SDK5.1) does not provide alternate (less confident) speech recognition results, and so is overly fragile. We continue to explore a more effective speech grammar definition permitting alternative recognitions, variations on user-interface issues, especially for correcting errors in conjunction with handwriting input.

9 Summary and Conclusion

This paper reports on work in progress on design. It does not include human-factors experiments. While such exercises are sometimes useful, we are not

convinced that the substantial effort to mount such an experiment would be worthwhile at this time: the reactions of a class of naive users may not be especially relevant, and would certainly delay the presentation of this material for others to consider.

We are still experimenting with the prospects of multimodal input of mathematics. We expect to see clear benefits of speech when saying "bold script capital R" or for distinguishing among the large number of symbols and collections of strokes that look essentially similar when written (recall `1<2` and `K2` as one example [11]). If speech can also be used for accurately conveying larger multi-token arrangements as indicated here, that represents a possible benefit. Speaking is "hands free" so that it can complement either a stylus or keyboard form of input.

While we do not expect speaking to be a "unimodal" mode of choice for very long expressions in a single gulp, we believe that in combination with pointing, speech can "fill in the boxes" which would be pointed-to and otherwise constructed or corrected via templates in an interactive input system.

This work in still in progress as of Summer, 2004, using speech tools from Microsoft (SDK 5.1 and SASDK 1.0/SDK5.2), handwriting tools from Microsoft and James Arvo[2].

10 Acknowledgments

This research was supported in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley. Thanks to Neil Soiffer for useful suggestions.

This work is being pursued (Summer, 2004) by a group including students C. Guy, S. Stanek, and M. Jurka supported by the NSF within the Research Experience for Undergraduates program and in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley.

References

[1] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover Publ. 1965.

[2] J. Arvo, http://www.cs.queensu.ca/drl/ffes/

[3] R. Fateman. voice+hand papers (drafts).

[4] R. Fateman. 2-D Display of Incomplete Mathematical Expressions

[5] R. Fateman. Boxes, Inkwells, Speech and Formulas

[6] C.A.R. Hoare. "An axiomatic basis for computer programming" *Comm. of the ACM 12* Issue 10 (October 1969) 576 —580 .

[7] N. Kajler, N. Soiffer. "A Survey of User Interfaces for Computer Algebra Systems." J Symbolic Computation 25(2): 127-159 (1998)

[8] D.E. Knuth. The TEXbook. Addison Wesley, 1984.

[9] Mathtalk http://www.metroplexvoice.com/

[10] MathML http://www.w3.org/Math/

[11] N. Matsakis http://www.ai.mit.edu/projects/natural-log/demo/

[12] D. Ragget, http://www.w3.org/People/Raggett/EzMath/

[13] T. V. Raman, AsTeR, *Auditory User Interfaces: Toward the Speaking Computer,* Kluwer Academic Publishers, Boston ISBN 0-7923-9984-6 August 1997, 168 pp. also http://www.cs.cornell.edu/Info/People/raman/aster/aster-toplevel.html

[14] Gerald Jay Sussman and Jack Wisdom with Meinhard E. Mayer, *Structure and Interpretation of Classical Mechanics,* MIT Press, 2001. online at http://mitpress.mit.edu/SICM/.

[15] M. Suzuki, http://infty.math.kyushu-u.ac.jp/index-e.html

[16] TEX Users Group http://www.tug.org/

[17] texmacs) J. van der Hoeven, TeXmacs http://texmacs.org

11 Appendix: Ambiguity, Syntax and Semantics

Humans tend to write ambiguous mathematics, expecting that the context, imposed by the human reader, will disambiguate.

For example if e_1 and e_2 are expressions, how do we parse arguments to functions like sin and cos? Does $\sin e_1 e_2$ mean

1. $(\sin e_1) \times e_2$ or
2. $\sin(e_1 \times e_2)$?

Please choose one and then examine the well-known equation displayed below.

$$\sin 2z = 2 \cos z \sin z$$

This is a formula *typeset exactly as given in a standard reference* (4.3.24 Abramowitz and Stegun [1]). You might read it out loud. Now note that on the left, $e_1 = 2$, $e_2 = z$ uses convention 2. On the right, $e_1 = z$, $e_2 = \sin z$ uses convention 1. Two different parsing conventions are used in the same equation. This is not unusual. Our point here is that linearizing the token stream is actually not sufficient to guarantee unambiguous syntax. (Additional rules about

the precedence of invisible multiplication between numbers and symbols can solve this particular problem, but there are others in which the writer and the typesetter conspire to confuse even the skilled reader.) For an amusing account of ambiguity in classical mechanics see the introduction to an on-line echanics book by Sussman and Wisdom [14] http://mitpress.mit.edu/SICM/book-Z-H-5.html.

On a semantic note, there are at least two proposals for a semantic encoding of mathematics, the most prominent of which is (the semantic component of) MathML [10]. On the face of it, writing a new paper in which one is using some speech or handwriting or keyboarding to produce a totally new notation makes it logically impossible to properly encode it with respect to its (up to this moment undefined) semantics. A kind of meta-language relating both new notation and the new semantics to that of existing notions is required. This meta-language too may have similar limitations, rather like explaining colors to a sightless person.

Where does this lead us? We are personally more inclined to first look for mappings of mathematical notations to some operational semantics such as computations in a given computer algebra system. Such a system generally imposes limits but within its realm of discourse is at least definite. Beyond that level, we may be forced to deal with a largely syntactic or geometric appearance of mathematics or aural representation!

For many purposes, including the obvious application of typesetting for consumption by mathematicians at a future time, this operational semantics is an additional boon, and can be combined with older material or material currently produced in a conventional manner. This traditional material has been limited to mathematical syntax encoded as typeset material, plus natural language commentary. Most current computer algebra system provide some kind of documentary framework or notebook which can handle such conventional material. The issue is how much further we can push in the semantics direction.