DEPARTMENT OF MATHEMATICS

TECHNICAL UNIVERSITY OF DENMARK

DTU

Interactive Mathematics Textbooks

Robert Sinclair

Mat-Report No. 1999–25

November 1999

Title of Paper:

Interactive Mathematics Textbooks

Author:

Robert Sinclair

Address:

Department of Mathematics
Matematiktorvet
Technical University of Denmark
Building 303
DK–2800 Kgs. Lyngby
DENMARK

Email:

`R.M.Sinclair@mat.dtu.dk`

Abstract:

An elementary mathematics textbook presupposes the existence of a language in which it may be written, some viewer which can perform the computations implied by interactivity, and an input language for the reader to use when editing mathematical expressions. We discuss the requirements interactive mathematical textbooks will make on the language in which they must be written, introduce a linear input grammar suitable for interactive input of simple mathematical expressions including vertical bars to denote absolute value, and present a prototype viewer written in Java which is capable of handling short texts without hyperlinks.

Key words: mathematics education, mathematical typesetting, interactive documents, symbolic computing.

1991 Mathematics Subject Classification: 68U15, 68T25.

1 Introduction

Experimentation is a vital part of the learning process in mathematics. A traditional textbook will provide many examples to allow the student to see the application of theory to concrete problems, but a tutor is required to answer questions such as “What happens if I change the 3 here to a 4?”.

While it is true that systems already exist which allow one to write interactive mathematics textbooks for school use (see, for example, [1, 2, 3, 4]), we claim that important considerations have been overlooked in the past. In particular, most previous work has concentrated on how to make use of pre-existing software in mathematics education, rather than first asking the more fundamental question of which requirements mathematics education puts on software, and then designing software to fulfil these requirements.

We will focus on elementary textbooks, whereas previous work in this area has focused on the senior levels of high school or introductory university courses.

Properties we believe an interactive mathematics textbook should have are

- **Correctness:** The statements made in the document should always be correct.
- **Interactivity:**
  - *(i)* It should be possible for the reader to change parts of the document and see the effects of these changes.
  - *(ii)* It should be possible to follow references within the document automatically.
- **Consistency:**
  - *(i)* It should not be possible to change parts of the document in a manner which interferes with correctness.
  - *(ii)* The notation used in the document and in interaction with it should be homogeneous.
  - *(iii)* The most elementary simplifications of the viewer software should not be more complex than those being explained in the text.
- **Distinction between Author and Reader:**
  - *(i)* The author is responsible for correctness and consistency.
  - *(ii)* The reader may change the document only within the framework set by the author.

Standard typesetting tools such as LaTeX [5] can be made to support interactive documents by using a viewer which can follow hyperlinks [6, 7], even to interactive visualization (such as [8]) or computational tools (such as [9]), but the reader is not able to make changes to the document itself.

```
The absolute value of q is written as
> abs(q);
                                        |q|
If we set
> q := 1/6^2 - 1/6*6;
                                          -35
                                     q := ---
                                          36
then its absolute value is
> abs(q);
                                        35
                                        --
                                        36
```

Figure 1: An excerpt of a sample interactive session with Maple VR5.1. Note the discrepancy between input and output notations (`abs(q)` and $|q|$, respectively). Furthermore, `1/6^2` is taken to mean $1/6^2$ whereas `1/6*6` is interpreted as $(1/6) \cdot 6$, a subtle syntactic difference which may lead to confusion in the mind of a reader who is not yet confident in arithmetic.

Standard computer algebra packages allow the reader to make changes to the document, but without any guarantee of consistency or author control, and without a homogeneous notation. They essentially allow an author to provide the reader with text, commands and the output of these commands. The system allows the reader (whom it cannot distinguish from the author, and this is the source of the problem) to alter the text or commands as they wish. Furthermore, the input and output notations are not homogeneous: one types in commands in a format modelled on typical programming languages but sees results in a format modelled on typical mathematical notation. See Figure 1. A final disadvantage of these packages is that they perform automatic simplifications which are hidden from the reader and often more complex than an elementary mathematics textbook would introduce. It could for example be difficult for a school student to realize that they actually entered the expression

$$\frac{1}{6^2} - \frac{1}{6} \cdot 6$$

in Figure 1 because it is automatically simplified to $-35/36$ before it is displayed. This makes it impossible to write a school text describing how to simplify such expressions using the computer algebra package as it stands. None of the computer algebra packages known to the author refrain from automatically simplifying x/x to 1 when the value of x is unknown, and some of them simplify to 1 even if x is known to have the value zero. Such automatic simplications are not only incorrect, they also make it, if not corrected, impossible to write a textbook covering the fact that

$$\frac{x}{x} = \begin{cases} 1, & (x \neq 0) \\ \text{undefined}, & (x = 0). \end{cases}$$

It seems to be a widespread belief that the vertical bars used to denote absolute value (as in $|-2| = 2$) cannot be incorporated into an standard parser for an equation editor, but we show that this is not necessarily true by defining a grammar which does include them. This makes it possible to come closer to fulfilling our aim of providing software which actually understands the notation students are being taught.

1.1 Scripts for Interactive Documents

The responsibility of the author for the correctness and consistency of an interactive document differs of course from what is required in the case of a standard book. It is useful to introduce the concept of a script (that which the author actually writes) as something distinct from the document the reader sees. A script will not only have text and equations which are intended for the reader to see, but also notes which are read by the software. These notes will partly be for purely formatting purposes, but also to specify the form the interactivity may take.

To give an idea of what a script may look like and what form the author's responsibility for correctness and consistency might take, consider the following poor example:

> We can use the rules given above to differentiate any polynomial such as $P = \underbrace{x^2+2001}_{\substack{Note: the\, reader\, may\, change\, this\\ but\, it\, must\, be\, a\, polynomial}}$ and differentiate it, getting $\underbrace{2x}_{Note: this\, must\, stay\, as\, it\, is}$. So we are finally in a position to differentiate Bloom's polynomial $\underbrace{17x^{17}+7x^7+7}_{Note: the\, reader\, may\, change\, this}$ ...

A reader would first see

> We can use the rules given above to differentiate any polynomial such as $P = x^2+2001$ and differentiate it, getting $2x$. So we are finally in a position to differentiate Bloom's polynomial $17x^{17}+7x^7+7$...

and be able to change the document to read

> We can use the rules given above to differentiate any polynomial such as $P = x+0$ and differentiate it, getting $2x$. So we are finally in a position to differentiate Bloom's polynomial $\sin z$...

but correctness is now lost. First, the derivative of $x+0$ is not $2x$, so the document is now incorrect. Second, "Bloom's polynomial" (assuming that this fictitious name only applies to the polynomial $17x^{17}+7x^7+7$) is no longer correctly displayed.

A better script is

> We can use the rules given above to differentiate any polynomial such as $P = \underbrace{x^2 + 2001}_{\substack{Note\text{: the reader may change this}\\ \text{but it must be a polynomial}}}$ and differentiate it, getting $\underbrace{2x}_{\substack{Note\text{: the reader may not change this, but}\\ \text{it must always be the derivative of } P}}$. So we are finally in a position to differentiate Bloom's polynomial $\underbrace{17x^{17} + 7x^7 + 7}_{Note\text{: the reader may not change this}}$ ...

The reader would see the same document at first. If he or she changes the definition of P, the software, following the author's notes, should also update its derivative. For example

> We can use the rules given above to differentiate any polynomial such as $P = x + 0$ and differentiate it, getting 1. So we are finally in a position to differentiate Bloom's polynomial $17x^{17} + 7x^7 + 7$...

Now it is clear that the software will not just have to typeset interactively, but also be able to compute, and this is what makes interactive mathematics textbooks difficult to implement. Of course, textbooks on chemistry, genetics or logic would also require some form of computation, but not necessarily of the same type.

A second type of notation a script will require is one corresponding to the `if/then/else` constructs of programming languages. One would like to be able to write scripts like

> Let there be $n = \underbrace{3}_{\substack{Note\text{: the reader may change this, but}\\ \text{it must always be a positive integer}}}$ rabbit$\underbrace{\text{s}\qquad}_{Note\text{: }\textbf{if}\ n>1}$ in a hat.

or

> Let us consider $f = \underbrace{\sin x}_{Note\text{: the reader may change this}}$, $\underbrace{\text{a function of } x.}_{Note\text{: }\textbf{if}\ f \text{ is a function of } x}$ $\underbrace{\text{which is just a constant.}}_{Note\text{: }\textbf{else if}\ f \text{ is a constant}}$ $\underbrace{\text{a function of variables other than } x.}_{Note\text{: }\textbf{else}}$

The latter could also be written (in a different scripting language, more like the one we have actually implemented in our prototype)

> Let us consider $f = \underbrace{\sin x}_{Note\text{: the reader may change this}}$, `if` f *is a function of* x `then` a function of x. `else if` f *is a constant* `then` which is just a constant. `else` a function of variables other than x. `endif endif`

but perhaps be most clearly expressed by writing

> Let us consider $f = \underbrace{\sin x}_{Note:\ the\ reader\ may\ change\ this}$,
> $$\left\{\begin{array}{ll} \text{a function of } x. & \textbf{if } f\ is\ a\ function\ of\ x \\ \text{which is just a constant.} & \textbf{else if } f\ is\ a\ constant \\ \text{a function of variables other than } x. & \textbf{else} \end{array}\right\}$$

but this (the choice of an elegant scripting language) is outside the scope of this paper.

There is also clearly a need for graphical functionality:

> The graph of the function $f = \underbrace{\sin x}_{Note:\ may\ change\ this}$ is $\boxed{\text{Note: plot the graph of } f \text{ here}}$.

where the graph should be able to be manipulated interactively by the reader.

Finally, there will be a need for hidden computations. For example, the poor script

> Let us consider $f = \underbrace{\sin x \cos x}_{Note:\ the\ reader\ may\ change\ this}$ $\underbrace{,\text{ or } a}_{\substack{Note:\ compute\ a=simplify(f)\ first \\ only\ show\ this\ if\ f\neq a}}$. Remember that $f + a = \underbrace{\sin x \cos x + a}_{Note:\ must\ be\ equal\ to\ f+a}$.

shows a simplified form of f only if it actually differs from f. The reader might first see

> Let us consider $f = \sin x \cos x$, or $\frac{1}{2}\sin 2x$. Remember that $f + a = \sin x \cos x + a$.

but could then alter the definition of f to $\sin \cos x$, which we will assume cannot be simplified further. The document might then read

> Let us consider $f = \sin \cos x$. Remember that $f + a = \sin \cos x + a$.

The example script is not entirely clear as it stands, because the variable name a appears in two different contexts. The scripting language would have to contain rules which define whether such a script contains two distinct variables with the same name, or just one. In the latter case the simplified value of f would appear in the sentence beginning with "Remember..." instead of the pronumeral a:

> Let us consider $f = \sin \cos x$. Remember that $f + \sin \cos x = \sin \cos x + \sin \cos x$.

Even if the script only made use of the variable name a in connection with simplifying f, there would still be the possibility that the reader might unwittingly use this name when changing an expression (for example, they might change $\sin x \cos x$ to $\sin ax \cos x$), and conflicts could still arise. So in an actual software implementation there would be a need for a clear distinction between hidden and public variables:

Let us consider $f = \underbrace{\sin x \cos x}_{Note:\ the\ reader\ may\ change\ this}$ $\underbrace{\text{, or } \tilde{a}}_{\substack{Note:\ compute\ \tilde{a}=simplify(f)\ first \\ only\ show\ this\ if\ f \neq \tilde{a}}}$. Remember that $f + a = \underbrace{\sin x \cos x + a}_{Note:\ must\ be\ equal\ to\ f+a}$.

Furthermore, it should not be possible for the reader to inadvertently use a "hidden" name. We will discuss this question again in connection with the prototype.

1.2 Hyperlinks

Links to other parts of a document (or to other material entirely) are a standard part of LaTeX, and clearly desirable in an interactive textbook. They are more difficult to define here, because the `if`/`then`/`else` constructs discussed in the previous section determine which parts of the script will be displayed. The consistency of the document would be compromised if a hyperlink were to point to a part of the document which does not exist for the reader. A mechanism is therefore required which allows the author to query the visibility of a hyperlink target. For example:

...and we conclude that $Q = 2$ $\underbrace{\text{(having used } \underbrace{\underline{\text{Lemma 12}}}_{Note:\ insert\ a\ hyperlink\ to\ Lemma\ 12}\text{)}}_{Note:\ \textbf{if}\ Lemma\ 12\ is\ visible}$.

Such a mechanism is however not without its dangers, since it does not guarantee the decidability of the visibility of a target. Take, for example

$\underbrace{\text{Is } \underbrace{\underline{\text{Brutus}}}_{Note:\ a\ hyperlink\ target} \text{ visible?}}_{Note:\ \textbf{if}\ Brutus\ is\ not\ visible}$

The simplest solution would be to discard such functionality and only allow hyperlinks to point to targets which are always visible. This is in fact not necessarily as drastic as it sounds, if hyperlinks are mostly used to refer to definitions and the like.

A more elegant test, which requires no more than static analysis of the structure of the text and therefore need only be performed once, is to demand that a hyperlink may only point to a target which must be visible if the hyperlink itself is visible, assuming that conditionals are uncorrelated. For example, all the underlined text in the following script must be visible if the hyperlink is visible:

$\underline{\cdots}$ $\texttt{if}_1$ $?_1$ $\texttt{then}_1$ $\underline{\cdots}$ $\texttt{if}_2$ $?_2$ $\texttt{then}_2$ $\cdots$
$\texttt{else}_2$ $\underline{\cdots}$ hyperlink $\underline{\cdots}$ $\texttt{endif}_2$ $\underline{\cdots}$
$\texttt{else}_1$ $\cdots$ $\texttt{endif}_1$
$\underline{\cdots}$ $\texttt{if}_3$ $?_3$ $\texttt{then}_3$ $\cdots$ $\texttt{else}_3$ $\cdots$ $\texttt{endif}_3$ $\underline{\cdots}$

(the fact that the hyperlink is visible means that $?_1$ must be true and $?_2$ must be false, but we are not able to imply anything about $?_3$).

2 Interactive Parsing of Mathematical Text

$\_$	$1\_$	$\frac{1}{\_}$	$\frac{1}{2\_}$	$\frac{1}{2}\_$	$\frac{1}{2}s\_$	$\frac{1}{2}si\_$	$\frac{1}{2}\sin\_$
$\frac{1}{2}\sin\_$	$\frac{1}{2}\sin^{2}\_$	$\frac{1}{2}\sin^2 x\_$	$\frac{1}{2}\frac{\sin^2 x}{\_}$	$\frac{1}{2}\frac{\sin^2 x}{\lvert\_\rvert}$	$\frac{1}{2}\frac{\sin^2 x}{\lvert x\_\rvert}$	$\frac{1}{2}\frac{\sin^2 x}{\lvert x\rvert\_}$	$\frac{1}{2}\frac{\sin^2 x}{\lvert x\rvert y\_}$

Figure 2: A series of snapshots taken from the screen of an equation editor using the grammar defined in the Appendix. Each box shows what is displayed after a new keystroke. The input string is `1/2 sin∧2x/|x|y`. Note that only standard mathematical notation appears on the screen.

An interactive textbook will need an equation editor which should respect "standard mathematical notation". There is unfortunately no standard formal definition for mathematical notation, although a subset of the notations used in mathematics (in particular the subset used in elementary textbooks or handbooks such as [10]) is understood and used almost universally, and it is that "standard mathematical notation" we will aim to approximate here. Since our aim is to formalize something which is only subjectively defined, we cannot avoid being subjective, and some decisions will of course exclude others, so that we can be fairly sure that what we are presenting here is only one of many self-consistent possibilities.

It is not entirely clear why mathematical software typically ignores standard mathematical notation, although [11] does provide some hints. For example, Maple [12] interprets `3(1+z)` as the constant 3, and `sin^3(z)` as $\sin^3$. If one wishes to enter $\sin^3(z)$ (i.e. with the meaning of equation 4.3.27 in [10]), then either `(sin^3)(z)` or `sin(z)^3` must be typed in. Mathematica [13] does interpret `3(1+z)` as $3(1+z)$, but `Sin^3[z]` as $\sin^{3[z]}$. `(Sin^3)[z]` is displayed as `Sin`^{3}`[z]` but does not have the same meaning as $\sin^3(z)$. Only `Sin[z]^3` is interpreted as $\sin^3(z)$. The environment described in [1] does address most of these issues by providing a buffer between the user and the computer algebra package Maple. Its syntax however has some unfortunate properties. Figure 3 of [1] illustrates the infix entry of the expression $2x^2+3(x-5)$. What is surprising about this illustration is that the actual input string is given to be

```
2*x^2   +3*x-5
```

This could easily mislead a school student into believing that the bracket around $x-5$ is unnecessary. It is also clear from the same figure in that paper that $|x|$ must be entered as `abs(x)` or `abs x`. The relevance of all of this to the subject of interactive textbooks is our requirement that the notation used in the document should be homogeneous. That is, input and display notations should not differ unnecessarily.

The EzMath notation ([14], but see [15] for a formal grammar definition) is the outcome of work at MIT and HP labs aimed at providing an easy to learn notation using plain text. It does allow the input of many expressions in a natural way (for example, `ax^2+bx+c` does represent $ax^2 + bx + c$), but the formal definition of its grammar reveals a number of weaknesses. The grammar accepts clearly meaningless input, such as

```
(,()) to the power of function 1+2((})
```

leaving much to semantic analysis. In practice, this would mean that an equation editor using this grammar must also have some knowledge of semantics, or accept such input (which could confuse a reader). Furthermore, it does not accept the standard notation $|x|$ for absolute value. This grammar certainly has the potential to be useful in connection with interactive mathematics textbooks, but we feel that it is less than optimal. It does not avoid the problem of having two notations, since it also has no provision for the notation $\sin^2 x$ (one must type in `(sin x)^2`).

Standard mathematical notation has a long history [16] and is in fact still changing. What we wish to stress here is that it is the result of a long period of evolution, and should therefore not be rejected out of hand. One of the advantages of standard mathematical notation is that the relevant mathematics is not buried in a sea of brackets. This is however also its primary disadvantage in the present context, because the lack of bracketing so easily leads to ambiguities. See [17] for a discussion of this point. Standard mathematical notation is inherently two-dimensional. It is now quite common to see a selection of templates associated with an equation editor (see for example [18]) because of the difficulties of entering expressions like

$$\frac{1+x}{1-x}$$

using linear (one-dimensional) input. What we wish to do in this section is derive a linear grammar which can cope with a useful subset of standard mathematical notation. Such a grammar could of course be used in conjunction with templates for expressions which cannot be written linearly.

The idea here is to mimic as much as possible the way one would read an expression out loud, or, if there is no obvious way to read out loud (e.g. $|x+||-z|+1||$), the way one would write it. For example, $2\sin x\cos x$ could be entered by typing `2sinxcosx`, and $|x+||-z|+1||$ by typing `|x+||-z|+1||`. At the same time, we wish to minimize the number of keystrokes necessary to input an expression, so we will be looking for a grammar which represents x^2 by something like `x^2` rather than something like `the value of the variable x squared`. We wish to distract a student as little as possible from the mathematics they are learning. We claim that it can be too much to expect a school student to master more than one system of notation while they are learning the concepts which define these notations. For example, it is not necessarily clear to a beginner that `Sin[x]^2` corresponds to what their teacher would write as $\sin^2 x$. It is even less likely that a beginner would be "fluent" in more than one notation. If they mistakenly type `Sin(x)^2` into a system which only recognizes `Sin[x]^2`, they will become confused or frustrated or both. We wish to avoid forcing a student to be more clear than the traditional textbooks they are otherwise learning from. This applies in particular to brackets. It would appear to be too much to demand that they enter `cos(sin(x))/sin(2)` for

$$\frac{\cos \sin x}{\sin 2},$$

since the brackets are not part of the mathematical notation.

The remainder of this section is devoted to an attempt to find a "natural" mathematical grammar which covers as large a subset of school mathematics as possible. This grammar will be specifically tailored to interactive input only, and therefore the emphasis will be on how it "sounds" rather than how it "looks". Due to the interactive context it is designed for, where a student will only see the input expression in standard mathematical notation (see Figure 2), it is more important for the grammar to "guess" what is meant than for it to be unambiguous. For example, if a student wants to enter

$$\frac{\sin x}{2},$$

and types `sinx/2`, they could see immediately if a grammar "misunderstood" it as

$$\sin \frac{x}{2}$$

and could take some action to ensure that the system understands what they actually want.

Since it is practical for development if such a grammar is also relatively easy to implement using common software tools, we have chosen to restrict ourselves to LALR(1) grammars (we will actually define an ambiguous grammar, but one with only shift/reduce conflicts which can be resolved: see Chapter 4 of [19]), such that the standard UNIX tools Lex and Yacc can be used for implementation. The Appendix includes a formal definition of the grammar in a form which is modelled on the Lex and Yacc input formats.

Despite recurring myths to the contrary, mathematical notation has never been entirely universal. The notation used here is inspired by continental European usage, where, in particular, multiplication is represented by a dot and the separator between integer and fractional parts of a decimal constant is represented by a comma. In this context, it makes sense to use the period key to enter the dot needed for multiplication. For example

$$\texttt{2.4,5} \quad \sim \quad 2 \cdot 4,5 \quad = \quad 2 \times \left(4 + \frac{1}{2}\right) \quad .$$

In line with LaTeX notation, the underscore (_) and hat (^) keys can be used to introduce sub- and superscripts respectively:

$$\texttt{x\_2\^{}b} \quad \sim \quad x_2^b \quad .$$

There is no reason why one could not also map the star key to represent multiplication, or up and down arrows to introduce sub- and superscripts, for example

$$\texttt{4*x}\!\downarrow\!\texttt{a}\!\uparrow\!\texttt{b} \quad \sim \quad \texttt{4.x\_2\^{}b} \quad \sim \quad 4x_2^b \quad .$$

These are implementation issues, which will not be discussed here any further.

In the case of school-level textbooks we can assume that variable names will always consist of only one letter, perhaps with an index. Incorporating this into the grammar allows a student to type `2ax`, just as they would write $2ax$ on paper (meaning $2 \cdot a \cdot x$, as in equation 3.3.18 of [10]).

Continuing in the same spirit, we have

$$\texttt{sin3xy} \quad \sim \quad \sin 3xy \quad \equiv \quad \sin(3 \cdot x \cdot y) \quad ,$$

and

$$\texttt{(x\^{}2+y)\^{}2=x\^{}4+2x\^{}2y+y\^{}2} \quad \sim \quad (x^2+y)^2 = x^4 + 2x^2y + y^2 \quad .$$

Note that we choose to accept only one atom (a number, a pronumeral, an absolute value in bars or a bracketed expression) as a power, otherwise `x^2y` would be interpreted as x^{2y}.

Not all mathematicians agree what the priorities in

$$2^{1^3}$$

actually are, and this is significant since $2^{(1^3)} = 2$ but $(2^1)^3 = 8$. We will therefore not incorporate strings like `2^1^3` into the grammar. Note that the notation $x \wedge 2$ for x^2 did not first arise in connection with the appearance of electronic computers, but dates back to 1845 (see Section 313 of [16]).

Difficulties can arise in connection with expressions like $\frac{1}{2}a$ or $\left|a^2|b|c\right|$. We will deal with the former case first. The problem is how to express the distinction between terms such as

$$\frac{2a}{3b} \quad \text{and} \quad 2\frac{a}{3}b \quad ,$$

without demanding brackets where they would not be used on paper. Both might be read out as “two a divided by three b”, but if one also pays attention to pauses, there is a possible distinction, since the second term might be read out as “two <pause> a divided by three <pause> b”.

The central idea is to use both spaces and brackets for grouping subexpressions, the spaces corresponding to pauses one would make when reading out an expression. For example,

$$\texttt{2a/3b} \quad \sim \quad \frac{2a}{3b} \quad ,$$

but

$$\texttt{2 a/3 b} \quad \sim \quad 2\frac{a}{3}b \quad .$$

Also

$$\texttt{sin2xy} \quad \sim \quad \sin(2{\cdot}x{\cdot}y) \quad ,$$

but

$$\texttt{sin2x y} \quad \sim \quad \sin(2{\cdot}x)\cdot y \quad .$$

Spaces are also useful as delimiters:

$$\texttt{sinhx} \quad \sim \quad \sinh(x)$$

but

$$\texttt{sin hx} \quad \sim \quad \sin(h \cdot x) \quad .$$

We therefore propose building a grammar around "pure products" (abbreviated to `pprod` in the Appendix), which are essentially products containing no spaces, such as

$$\texttt{2a\_3b\^{}7(x-7)zq} \quad \sim \quad \texttt{2a\_3b\^{}7(x-7)z.q} \quad \sim \quad 2a_3b^7(x-7)zq \quad ,$$

quotients (`quot`) of these, such as

$$\texttt{2a\_3b\^{}7/(x-7)zq} \quad \sim \quad \frac{2a_3b^7}{(x-7)zq} \quad ,$$

and products (`prod`) of those, formed using spaces, such as

$$\texttt{2a\_3 b\^{}7/(x-7)z q} \quad \sim \quad 2a_3\frac{b^7}{(x-7)z}q \quad ,$$

which then appear in sums and differences:

$$\texttt{2a\_3+b\^{}7/(x-7) z-q} \quad \sim \quad 2a_3 + \frac{b^7}{(x-7)}z - q \quad .$$

In some cases it is not possible to avoid brackets. One cannot enter

$$\frac{1+x}{1-x} \quad \text{or} \quad z^{1-b}$$

as linear strings without using brackets. It may appear that again using spaces, as in `1+x / 1-x`, may solve the problem, but such a solution leads to confusing situations such as `1+x / 1-1/2 x`. Is that

$$\frac{1+x}{1-\frac{1}{2}}x \quad ,$$

$$\frac{1+x}{\left(1-\frac{1}{2}\right)\cdot x}$$

or

$$\frac{1+x}{1-\frac{1}{2}x} \quad ?$$

A partial solution is to expect input like `(1+x)/(1-x)`, but only display the brackets during input of that ratio. Of course an interactive equation editor could (and most do) offer templates for such situations. The user first selects a template (something like $\frac{\square}{\square}$), and then enters text into the boxes. This process cannot be described in the context of a traditional grammar, so we will not discuss it further here even though it is clearly a desirable feature to have in an interactive mathematics textbook. In any case, there is a need for a linear grammar to interpret what is typed into a template.

Functions offer the next series of problems.

First, one usually understands

$$f(1+x)$$

to represent the value of the function f given the argument $1+x$, but

$$x(1+f)$$

to represent the product of the quantity x with $1+f$. That is, certain letters have certain traditional connotations. Although one can indeed base a grammar around traditional connotations (Fortran 77's implicit typing is an example of this – see Section 7.2 of [20]), it would be better in the context of educational material (where the reader may not be aware of these traditions) to require that the distinction between variable and function names be made in the script. For example, one could demand that function names be declared at the beginning of a script.

Second, it is not immediately clear whether `f(1,2)` means $f(1+2/10)$ or $f(1,\ 2)$ (i.e. whether there is one decimal or two integer arguments). In a mathematical text, one would usually define a function to be of one or two arguments, and that would define the notation.

This means that our "natural mathematical grammar" must be context specific. In this paper, we will concentrate on the context-free subset of such a grammar, since that already suffices for much low-level mathematics. Furthermore, all the common elementary functions can simply be encoded into a context-free grammar, and that is what we will do.

The common elementary functions exp and ln (or log – we are only interested in grammatical aspects here, and will for that reason also ignore the notation e^x in this paragraph) are usually written as

$$\exp 2y \qquad \text{or} \qquad \exp\left(1+\frac{1}{1-q}\right) \quad .$$

Equation 4.2.10 in [10] includes both possibilities:

$$\exp(z \ \ln a) \quad .$$

That is, the brackets are optional. Furthermore, one also commonly finds expressions such as $\exp\, n(1+z)$, meaning $\exp[n(1+z)]$. These notations fit well into the framework already presented, using "pure products" to specify the scope of a function's argument. So we have

$$\texttt{expx} \quad \sim \quad \texttt{exp(x)} \quad \sim \quad \exp x$$

and

$$\texttt{expn(1-z) q} \qquad \sim \qquad \exp[n\cdot(1-z)]\cdot q \quad .$$

It is however not quite so simple. One could also write (see [17])

$$\frac{1}{2}\left[1+\frac{\sin(n+\frac{1}{2})x}{\sin\frac{1}{2}x}\right] \quad \text{meaning} \quad \frac{1}{2}\left[1+\frac{\sin\left\{(n+\frac{1}{2})x\right\}}{\sin\frac{1}{2}x}\right]$$

or

$$\exp(1+x)z \quad \text{meaning} \quad z\exp(1+x) \quad .$$

We assume that the latter style is more common in mathematics (see, for example, equation 15.1.15 in [10], where the author has felt the need to set square brackets in $\sin[(2a-1)z]$), and restrict those "pure products" which can be arguments of elementary functions to be either a bracket or a "pure product" not beginning with a bracket. In the Appendix, a "pure product" not beginning with a bracket (and there will be further restrictions made below) is denoted `fpprod` (the `f` indicating its association with functions).

The fact that the common elementary functions have names of more than one letter can be a cause of minor irritation, if one for example enters $x^{\sin x}$, since `x^si` will be understood as $x^s \cdot i$ with the i not belonging to the exponent, but as soon as the `n` is entered the i jumps up, because now it can be seen to belong to a function name. There does not seem to be an elegant way of avoiding such jumps. Defining `x^si` to mean $x^{s\cdot i}$ will only conflict with the interpretation of `x^2y^2` (which we want to mean x^2y^2 and not x^{2y^2}).

$\sin z_1 \cos z_2$ is quite common mathematical notation for $(\sin z_1)\cdot(\cos z_2)$ (as in equation 4.3.33 in [10]; note also the brackets in equation 9.1.42 in [10]: $\cos(z\ \sin\theta)$), and yet $\sin 2z$ usually means $\sin(2z)$ (as in equation 4.3.24 in [10]). One also has expressions such as the products on the right hand side of

$$[\cos\theta \quad \sin\theta]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \cos\theta\ \cdot\ x_1 + \sin\theta\ \cdot\ x_2 \quad .$$

This forces us to refine the definition of a "pure product" in the context of the argument of a function yet again. We restrict an `fpprod` to be a product without spaces which does not begin with a bracket, and does not include a factor which is itself a function. This results in

$$\texttt{sin2xcos2x y} \quad \sim \quad (\sin 2x)\cdot(\cos 2x)\cdot y \quad ,$$

but

$$\texttt{sin(2xcos2x)y} \quad \sim \quad \sin(2x\cdot\cos 2x)\cdot y \quad ,$$

$$\texttt{sin2(1+q)y} \quad \sim \quad \sin\left[2\cdot(1+q)\cdot y\right]$$

and

$$\texttt{sin2(cosq)y} \quad \sim \quad \sin\left[2\cdot(\cos q)\cdot y\right] \quad .$$

The fact that we are only interested in LALR(1) grammars introduces some restrictions at this point. The question is how to handle input such as `sinx.(1+5.cosx)`. An LALR(1) parser will have shifted `sinx.(1+5.` to its stack before it can see the cosine, but it is too late at this point to reduce the `sinx` because an LALR(1) parser can only recognize handles at the top of the stack. For this reason, an `fpprod` may not involve any products written using a period, or quotients built using a `/`. A disadvantage of this rule is that we have

$$\texttt{sin x/2} \quad \sim \quad \frac{\sin x}{2}$$

instead of $\sin(x/2)$, but on the other hand we do have (equation 4.1.18 in [10])

$$\texttt{ln z/ln a} \quad \sim \quad \frac{\ln z}{\ln a} \quad .$$

A possible solution would be to change the grammar to allow `sin xy/2` to mean $\sin(xy/2)$ but leaving `sinxy/2` to mean $(\sin xy)/2$. While this does appear attractive, it is difficult to apply the same reasoning to the pair of strings `sinhx/2` and `sin hx/2`, since the blank plays a quite different role in this context (that of clarifying whether sin or sinh is meant).

The notation $\sin^{17} x \equiv (\sin x)^{17}$ is no particular problem:

$$\texttt{sin\^{}17x} \quad \sim \quad \texttt{sin\^{}17 x} \quad \sim \quad \sin^{17} x$$

(note that this is an improvement over LaTeX, where `\sin^17x` appears as $\sin^1 7x$) and

$$\texttt{sin\^{}17 34x} \quad \sim \quad \sin^{17} 34x \quad .$$

It would have been possible to support the notation `sin(x)^2`, but this notation actually only rarely appears in mathematics textbooks. The fact that this notation is widespread amongst computer languages and computer algebra packages is no valid reason to consider it here. We have chosen not to support the notation $\sin^{-1} x$ for $\arcsin x$.

Factorials can also be treated quite simply. We have chosen to let them bind more strongly than any other operator (except for variable name indexing):

$$\texttt{3!\^{}4!+U\_2!} \quad \sim \quad 3!^{4!} + U_2! \quad .$$

The symbol $\sqrt{}$ presents us with a new difficulty, because no key on a standard keyboard resembles it. One could follow standard computer language syntax in this case, and use the function name `sqrt`, so that `sqrtx` would mean $\sqrt{x}$. The name `sqrt` however does not allow a simple extension to cope with cube roots like

$$\sqrt[3]{8}$$

because `sq` is actually an abbreviation for "square", and `sqrt^3 8` could just as easily mean $(\sqrt{8})^3$. In LaTeX, one uses the notation `\sqrt[3]{8}`, but this could cause conflicts with expressions such as $\sqrt{3}\{8\}$. Since we are already using `^` to indicate that something should be raised, it might be better to write $\sqrt[3]{8}$ as `^3sqrt8`, but then there will be ambiguities in connection with inputs such as `2^3sqrt8`. An entirely different possibility would be to use the notations

$$\texttt{\\/8} \quad \sim \quad \sqrt{8}$$

and

$$\texttt{\\3/8} \quad \sim \quad \sqrt[3]{8} \quad ,$$

where the weak visual similarity between `\/` and $\sqrt{}$ could be an advantage. In short, there does not seem to be a simple way of encoding the various roots. Square roots of sums or differences need to be treated with some additional mechanism like a template, or by expecting the reader to enter brackets which are only shown during input of the radical. For example, `\/(1+x)` would be displayed as $\sqrt{(1+x)}$ during input, but otherwise $\sqrt{1+x}$. It is interesting to note that, as late as 1915, the Council of the London Mathematical Society recommended $\surd(ax^2+2bx+c)$ in place of $\sqrt{ax^2+2bx+c}$ (see Section 334 of [16]).

The greatest difficulty of all is posed by the vertical bars commonly used to denote the absolute value of an expression. When writing on paper, the height of these bars can be used to clarify which pairing is meant if the expression would otherwise be ambiguous. For example, we find

$$\Big| |z_1| - |z_2| \Big|$$

in the inequality 3.7.29 in [10]. Furthermore, one could write

$$\Big| x + 3|y| - z \Big| \quad \text{for} \quad |x + (3 \cdot |y|) - z|$$

and

$$|x + 3|y\Big| -z \Big| \quad \text{for} \quad (|x+3|) \cdot y \cdot (|-z|) \quad .$$

A standard keyboard will not allow a user to express these differences in height, so an entry such as `|x+3|y|-z|` is simply ambiguous. From a grammatical point of view, the ambiguity lies in the combination `3|`, since the bar could either be a left bar (as in $\underline{3|}1-z|$) or a right bar (as in $z|1-\underline{3|}$). Recall that the grammar being described here is intended to be applied in an interactive context. Suppose that the input so far consisted of `3|4|`. Should this be interpreted as $3(|4|)$ or the beginning of $3\cdot(|4\cdot(|...$? An naive grammar must choose the latter, but this leads to the unsatisfactory situation that no vertical bar can ever be parsed as a right bar. This becomes clear when one realizes that the second vertical bar in combinations such as `||` (assuming no previous left bar, as in $1+||2-z|-x|$) or `|!|`, or any bar after a binary infix or unary prefix operator (i.e. `/|` or `+|`) must be a left bar. One solution would be to always interpret a vertical bar following an atom as a right bar. In such a grammar, `|1+|1-z||` would be correctly interpreted as $|1+(|1-z|)|$, but `1+3|1-z|` could not be parsed unless one explicitly put a dot in: `1+3.|1-z|`. This is indeed the solution taken in the "inner grammar" presented in the Appendix, but is much too radical. Such a grammar will not accept inputs such as `3|x|`

The solution we propose is to introduce an "outer grammar" in which juxtaposition of an atom with a bar (e.g. `3|`) does imply multiplication, but to switch to the "inner grammar" inside a pair of bars. The parser will always begin in "outer" mode, and also return to "outer" mode when analyzing the contents of brackets, but enter the "inner mode" after each left bar. So, for example,

$$\texttt{1+3|1-z|} \quad \sim \quad 1+3\cdot|1-z|$$

because the `3|` is interpreted by the "outer grammar" as $3\cdot|$, whereas the `z|` cannot be parsed by the "inner grammar", so the "inner grammar" parses the atom z and returns to the "outer grammar", which is waiting for a right bar. The "inner grammar" essentially rejects any bar which could be interpreted as a right bar, effectively ending the absolute value expression as soon as possible. So, we have

$$\texttt{||z\_1|-|z\_2||} \quad \sim \quad \Big||z_1|-|z_2|\Big| \quad ,$$

$$\texttt{|x+3|y|-z|} \quad \sim \quad (|x+3|)\cdot y\cdot(|-z|)$$

and

$$\texttt{|x+3.|y|-z|} \quad \sim \quad |x+(3\cdot|y|)-z| \quad .$$

The "inner grammar" must be modified significantly with respect to the "outer grammar" when it comes to input such as `|sin|x...` or `|1/|2...`, where the second bar in each case must be a left bar. So we have

$$\texttt{|sin|x-2|+1|} \quad \sim \quad |\sin(|x-2|)+1|$$

and

$$\texttt{|1/|2-1/|p||q|} \quad \sim \quad \left|\frac{1}{\left|2-\frac{1}{|p|}\right|\cdot q}\right|$$

but also

$$\texttt{|sin9|x-2|+1|} \quad \sim \quad \left\{ |\sin 9| \cdot x \right\} - 2 \cdot |{+1}|$$

and

$$\texttt{|1/|2-1/a\^{}2|p||q|} \quad \sim \quad \left| \frac{1}{\left|2 - \frac{1}{a^2}\right| \cdot p} \right| \cdot |q| \quad .$$

The common notation

$$2\frac{3}{4} \equiv 2 + \frac{3}{4}$$

is extremely difficult to define in a generally meaningful way, since one can so easily construct examples which are confusing. Take for example

$$2\frac{3}{4}2, \quad 2\frac{(3)}{4}, \quad 2\frac{1+2}{4}, \quad \text{or} \quad 2\frac{a}{4} \quad .$$

It is however generally accepted. We will define it in a rather strict manner here (conforming to standard mathematical practice), and demand that the three quantities appearing all be positive integers. That is, $2\frac{-3}{4}$, $a\frac{3}{4}$ and $2\frac{1+2}{4}$ will all be considered to be products. To be consistent with what has already been discussed, we will use spaces to separate the integer and fractional parts:

$$\texttt{34 56/78} \quad \sim \quad 34\tfrac{56}{78} \quad .$$

We cannot, however, include it in an LALR(1) grammar because of the amount of lookahead required to handle both `2 3/4` and `2 3/a`. Take the example of `ln 2 3/a`. An LALR(1) parser will already have shifted `ln 2 3/` onto the stack before it sees that the string means $\ln(2) \cdot [3/a]$ instead of $\ln(2 + 3/a)$, but then the handle corresponding to $\ln(2)$ is no longer at the top of the stack. Similar problems arise with partial inputs such as `cos ln 2 3/` or `2.3 1/`. Recognition of this notation must therefore either occur during lexical or semantic analysis. Neither alternative fits in perfectly with the constructs already discussed. Recognition during lexical analysis is simplest to define when using a compiler generator, and the Lex and Yacc specifications in the Appendix implement it. A new token is introduced (named `JUX`, because it corresponds to the juxtaposition of an integer with a fraction). The disadvantage of this solution is that it conflicts with the use of spaces introduced above. For example, we now have

$$\texttt{34 56/7a 9b} \quad \sim \quad 34\tfrac{56}{7}{\cdot}a{\cdot}9{\cdot}b$$

and

$$\texttt{sin2 1/2+cos2 1/b} \quad \sim \quad \sin\left(2 + \tfrac{1}{2}\right) + \cos(2) \cdot \tfrac{1}{b} \quad .$$

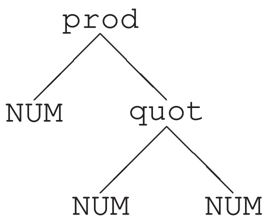
We also have

$$\texttt{34 56/7/9} \quad \sim \quad \frac{34\frac{56}{7}}{9} \quad .$$

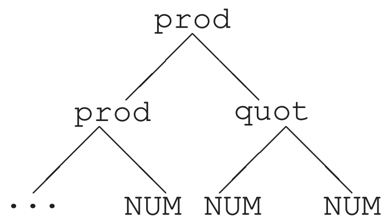
We must however remember that the input strings are not meant to be seen by the document's reader, so the actual effect of these conflicts on user-friendliness may be less than it would appear when one looks at the above examples. Also note that

$$\texttt{3 1/2!} \quad \sim \quad \left(3+\tfrac{1}{2}\right)! \quad .$$

The alternative (seen from an abstract point of view) is to wait until after the parse tree has been constructed, and then look for branches of the forms



or



(where we have used names consistent with the "inner grammar" of the Appendix, but only shown nontrivial productions). This latter method is suited to hand-written recursive descent parsers, since all that is required is a test for such forms inside the routine which parses products (`prod`) and to change the parse tree as it is being constructed. This alternative solution does respect the spirit of the use of spaces introduced above: `3 1/2 a` is interpreted as $(3+1/2)\cdot a$ but `3 1/2a` is interpreted as $3\cdot(1/[2a])$. Functions cause some minor problems of interpretation (`sin 2 1/2` is interpreted as $\sin(2)\cdot(1/2)$). Although this behaviour could be changed by also recognizing branches containing function applications, one would actually need an infinite number of such checks corresponding to expressions such as $\sin\cos 2\frac{1}{2}$, $\log\sin\sinh 2\frac{1}{2}$ etc., so it would seem better not to try. Finally, note that `4 2/3!` would be parsed as $4\cdot(2/[3!])$.

2.1 Special Considerations during Input

Here we will be concerned with the insertion of a cursor, pairing of brackets and the use of temporary brackets during interactive input.

A cursor is extremely useful as an aid in predicting where the next character will appear. The use of blanks to define groupings introduced above provides a case in point. The partial input `1/2` can be displayed as

$$\frac{1}{2\_} \quad ,$$

changing to

$$\frac{1}{2}{}^{-} \qquad \text{and then} \qquad \frac{1}{2}a\_$$

if followed by a blank and an `a`, or

$$\frac{1}{2a\_}$$

if just followed by an `a`.

Automatic prediction is however not always possible in the context of the grammar introduced above. For example, the partial input `x^2` could be followed by a `1` or a `y`, which would be interpreted as x^{21} or x^2y respectively. It is therefore not possible to decide whether to display $x^{2\_}$ or $x^2\_$ on the basis of the partial input.

A solution which would appear to be acceptable in most situations is to assume that a digit will most probably be followed by another digit (i.e. display $x^{2\_}$ for the partial input `x^2`). One could perhaps continue in this spirit by assuming that a combination of characters which could be the beginning of a function name is in fact just that, so that the partial input x^l would be displayed as $x^{l\_}$, in the expectation that an `n` will follow. This will however not work with names of more than two characters, since the LALR(1) grammar introduced above must parse the partial input `x^si` as $x^s i$.

It is not obvious how to introduce the concept of a cursor into the grammar presented in the Appendix, particularly since only one cursor should be displayed at a time. A partial input such as `2(a^3` should be displayed as $2(a^{3\_})$ and not as $2(a^{3\_}\_)\_$ (the last two correspond to the contents of the bracket and the entire input expression, both of which could be understood as being incomplete in the absence of an end of line character).

The automatic pairing of brackets during input is a similar aid to input, but which introduces no implementation difficulties. For example, the partial input `2(x/[8|-u^z` can be displayed as

$$2\left(\frac{x}{[8|-u^{z\_}|]}\right) \quad .$$

Subsequently entering | changes the display to

$$2\left(\frac{x}{[8|-u^{z}|\_]}\right)$$

and so on (see also Figure 2).

Note that the automatic pairing of brackets can be handled by providing extra productions lacking right brackets to the grammar presented in the Appendix. The same applies to the bars used to denote absolute value. For example, we can redefine the production for `abs` to be

```
abs:  '|' exp '|'          |exp|
|     '|' exp              |exp|
;
```

The resulting shift/reduce conflicts are resolved in favour of shift by Yacc, and this is the behaviour we are looking for.

As was already discussed in connection with the use of templates to enter expressions such as $(1+x)/(1-x)$, one can make use of temporary brackets, which are input by the reader but only shown while they are necessary. One must however be careful when defining “necessary” in this context. While it is clear that the partial input `(1+x)/(1-x)+z^(1+` could be displayed as

$$\frac{1+x}{1-x} + z^{(1+\_)} \quad ,$$

`(1/x)/[y/z]` should probably be left as

$$\frac{\left(\frac{1}{x}\right)}{\left[\frac{y}{z}\right]} \quad \text{rather than} \quad \frac{\frac{1}{x}}{\frac{y}{z}} \quad .$$

A final consideration is the use of spaces or dots to improve the clarity of a displayed expression. For example, our grammar allows us to enter $4! = 1 \cdot 2 \cdot 3 \cdot 4$ as `1 2 3 4`, but the display

$$1\,2\,3\,4$$

could mislead the reader into believing that this is the single number 1234, particularly if they are returning to a page they cannot remember altering. It would therefore seem wise to introduce either larger spaces or dots automatically:

$$1 \cdot 2 \cdot 3 \cdot 4 \quad .$$

The same applies to certain products like

$$\sin 2x\ y\ \ln z \quad .$$

Expressions such as

$$\cos\theta\, x_1 + \sin\theta\, x_2$$

may also require separating dots, or at least some extra spacing (otherwise they could look like $\cos(\theta x_1)$ etc.).

2.2 Automatic Completion of Expressions

It makes little sense to punish a reader of an interactive book who forgets to close a bracket before pressing RETURN/ENTER, particularly if automatic pairing is implemented. For example, the partial input `sin(x` would be displayed as

$$\sin(x\_) \quad .$$

If the reader presses RETURN/ENTER at this point, one is only being reasonable in assuming that they most likely meant to input

$$\sin(x) \quad .$$

More serious cases such as a missing argument for an elementary function are best handled by assuming an appropriate value, rather than producing an error message. For example, the input `2+sin` can be completed to $2 + \sin 0$. Such naive rules based on syntax alone cannot avoid completions which are not mathematically defined. If we complete an empty bracket by assuming some constant such as 0 or 1, then we cannot avoid unfortunate completions such as `0/(` becoming

$$\frac{0}{(0)}$$

or `0/(sin^2x+cos^2x-[` becoming

$$\frac{0}{(\sin^2 x + \cos^2 x - [1])}$$

respectively (neither of which are defined).

Despite the possiblity of unfortunate cases, automatic completion would appear to be a desirable feature in an interactive equation editor. For example, to implement this for absolute values, we can yet again redefine the production for `abs` to be

```
abs:  '|' exp '|'           |exp|
 |    '|' exp               |exp|
 |    '|'                   |0|
 ;
```

The same applies to brackets. The resulting shift/reduce conflicts are resolved correctly in favour of shift by Yacc.

2.3 On Using Supersets of LALR(1) Grammars

Our decision to restrict ourselves to an LALR(1) grammar was purely due to technical considerations. Given that we do not expect long input strings from a reader (even 100 characters could be said to be a lot to type in), it is not clear why we could not drop this restriction and move to a larger class of bottom-up grammars which can for example recognize inputs such as `sinx.(1/[1+cosy])` as a product of two subexpressions each containing an elementary function application, and parse it for this reason (instead of just using the dot to end the argument of the sine) as

$$\sin(x) \cdot \frac{1}{1 + \cos y}$$

rather than

$$\sin\left\{x \cdot \frac{1}{1 + \cos y}\right\} \quad .$$

The disadvantage of such a more flexible approach is that it is even more liable to produce jumps in the display as characters are input as is the LALR(1) grammar already presented.

In the example above, the partial input `sinx.(1/[1+co` would probably be displayed as

$$\sin\ x\cdot\left(\frac{1}{[1+co\_]}\right) \quad ,$$

but entering `s` would cause a discontinuous jump in this display to

$$\sin x\ \cdot\ \left(\frac{1}{[1+\cos\_]}\right) \quad .$$

In the case of the partial input `x^si`, defining a grammar which assumes that an `n` will follow will in fact avoid the jump associated with parsing and displaying this input as $x^s i\_$ but then $x^{\sin}\_$, but will still result in a jump from $x^{si}\_$ to $x^s iq\_$ if the next character is in fact a `q`.

It would in fact appear as if every single advantage of a more general parser would come with the penalty of as many or yet more such jumps. Or, to put it another way, an LALR grammar appears to minimize the possibility of jumps (in fact, an LALR(0) grammar would probably be optimal in this respect).

3 Desirable Properties for the Computing Engine

It is obvious that the viewer software will have to be able to compute, and it is tempting to consider the possibility of using a standard computer algebra package as the computing engine (as indeed was done in [1], where Maple was used). Here we run into problems of both technical and pedagogical nature.

The technical problems which are most serious are those which arise if one desires portability. It is never a simple matter to link two software components in a portable way. The fact that the system described in [1] only runs on Macintosh computers is a case in point. The fact that the syntax of standard computer algebra packages changes with time adds a further serious difficulty.

Standard computer algebra packages are designed to solve large problems for "experts". An "expert" is someone who can recognize an incorrect result and alter the input accordingly, whereas a typical school student will question their own knowledge if confronted with a suprising result, and perhaps conclude that they had misunderstood the mathematics. The fact that computer algebra packages are designed to solve large problems (i.e. they use algorithms which have low computational complexity rather than what students are taught in school) means that they are often incapable of solving "simpler" ones without extra help. For example, no standard computer algebra package will factorize $x^2 - 13$ unless the user provides further information (that $\sqrt{13}$ may appear in the answer) or makes use of special knowledge about other commands (for example, that solving $x^2 - 13 = 0$ will provide a partial solution), and yet this is a typical example of what every school student is expected to be able to do. Many of these problems are discussed and solutions given in [1]. In particular, it is possible to provide one's own routines which circumvent most of the problems of the package's built-in commands. There are, however problems associated with elementary mathematics which cannot be solved so simply. These are related to our demand that the most elementary simplifications of the viewer software should not be more complex than those being explained in the text. For example, if one wishes to discuss commutativity of addition and multiplication, one runs into the problem of automatic rearrangement of expressions in Maple. Both $a \cdot b$ and $b \cdot a$ are immediately displayed as ab. One could make use of "neutral operators", and have the front-end viewing software pass `a&**b` and `b&**a` on to Maple (for $a \cdot b$ and $b \cdot a$ respectively). The same mechanism will also prevent automatic simplifications such as $x \cdot 1$ becoming x. The front-end could of course perform the reverse transformation when displaying. This can also be done in connection with addition. One would therefore hope that one could use the mechanism of neutral operators to perform explicit simplifications such as

$$\begin{aligned} 3 \cdot a + a \cdot 3 \cdot 1 &= 3 \cdot a + a \cdot 3 \\ &= 3 \cdot a + 3 \cdot a \\ &= (3+3) \cdot a, \end{aligned}$$

but Maple defines the precedence of neutral operators to be equal (and this cannot be changed), and `3&**a&+a&**3&**1` is understood as `(((3&**a)&+a)&**3)&**1`. What should be clear by now is that one is faced with what amounts to rewriting the most basic operations of Maple. Given the technical problems already hinted at above, there is no longer any obvious advantage in using such a computing engine.

The alternative is to include the computational engine in the viewing software, which of course implies that this engine must be written anew. This does sound like a case of reinventing the wheel, but it should be apparent that the properties the computational engine for an elementary mathematics textbook should have are in fact quite far removed from those a commercial computer algebra package must have, and the set of problems to solve much more restricted (i.e. restricted to elementary mathematics).

4 A Prototype Implementation

The prototype viewing software has been written in Java [21] (version 1.1.7) for portability. We will not provide implementation details here since they are of secondary importance, and guided by performance issues which may be specific to the Java compiler we are using (for example, passing the script to the Java viewing software via a parameter in the APPLET tag in HTML [22] would appear to be too time costly). Many relevant implementation issues have already been discussed in [23]. The prototype's address is `http://www.mat.dtu.dk/persons/Sinclair_Robert_Michael/VIDIGEO/IntMatTex.html`.

4.1 The Prototype Scripting Language

What we will concentrate on is the scripting language used in the prototype. This is prototypical in itself. While it would seem to be an adequate basis for further work from the point of view that it allows one to write interactive documents, it lacks the readability an author needs to be able to keep an overview of their own work. The question of the elegance of the scripting language is outside the scope of this paper, but would be without doubt of great importance to non-expert authors.

This Section should be read in conjunction with sample scripts A and B in the Appendix.

Tokens in the scripting language are separated by white space. A token beginning with a backslash character is understood as a command, all other tokens as either words, punctuation or portions of mathematical expressions. A word or punctuation is simply printed at the current position on the screen in the current font. The commands `\plain`, `\italic`, `\bold`, `\normalsize`, `\largesize` and `\smallsize` change the current font. The commands `\endofline`, `\skipline` and `\newparagraph` change the current cursor position. The similarities with LaTeX should be obvious. The prototype viewing software is indeed a quite primitive mathematical typesetter. Much of that functionality could in principle be taken over by a browser capable of displaying MathML[24].

The command pair `\literalmaths`/`\endmaths` is used to display mathematical expressions exactly as they stand. For example,

```
\literalmaths 1+x \endmaths
```

results in $1+x$ being displayed, regardless of the value of x.

The command pair `\showmaths`/`\endmaths` performs substitutions before displaying a mathematical expression. For example,

```
\showmaths 1-x \endmaths
```

results in $1-t^s$ being displayed if x has been given the value t^s at a point in the text preceding this, or $1-(17+y)$ being displayed if x has been given the value $17+y$. Note the necessity of automatically introducing brackets.

The command pair `\hidemaths`/`\endmaths` assigns values to variables without displaying anything. For example,

```
\hidemaths Q=|1-z| \endmaths
```

results in Q being assigned the value $|1-z|$.

The command pair `\inputmaths`/`\endmaths` assigns values to variables and displays the assignment. Furthermore, it allows the reader to alter the right hand side of the assignment. The expression given in the script is merely the expression assigned and displayed until the reader makes changes to it. For example,

```
\inputmaths Q=|1-z| \endmaths
```

results in Q being assigned the value $|1 - z|$ and

$$Q = |1 - z|$$

being displayed. If the reader changes $|1 - z|$ to $13b$ then Q is also assigned the value $13b$, and all subsequent uses of Q are updated.

The command pair `\2dplot/\end2dplot` creates and displays an interactive graph of a function. The syntax is a sequence of three expressions, separated by commas. The first expression must be the function (prefixed by a dollar sign) applied to the independent variable. The final two expressions are the limits of the domain of the independent variable. For example,

```
\2dplot $f(x), 0, 1 \end2dplot
```

results in the graph of $f(x)$ being displayed for $x \in [0, 1]$. The graph is interactive in the sense that the reader can use the mouse to define rectangular regions to zoom in on, or reset to the original domain and range by just clicking.

The command pairs `\literalmaths / \endmaths`, `\showmaths / \endmaths`, `\hidemaths / \endmaths`, `\inputmaths / \endmaths` and `\2dplot / \end2dplot` each enclose one or more non-command tokens which are appended (including their separating white space) and parsed according to a grammar slightly extended with respect to the one given in the Appendix. The first extension to the grammar is the introduction of the dollar sign as an escape character to prefix variable names of more than one character and function names. The point of this extension is to allow certain variables and "system functions" to be hidden from a reader. Any escaped name may contain more than one character. If such a name is followed immediately by a right bracket ("`(`") it is understood to be a function name. The parser then reads a list of arguments (each of them an expression) separated by commas and ended by a right bracket ("`)`"). For example, `$zz+$yy` represents the sum of the variables with names `zz` and `yy`. `$P(x)` represents the function `P` applied to `x`. `P(x)` without the dollar represents the product $P \cdot (x)$. `$now(a,b+17,x^2/$P(x))` represents a call to the function named `now` with three arguments, the third being $x^2/P(x)$. Note that certain elementary mathematical functions are "hardwired" into the grammar presented in the Appendix, so `sin(x)` represents $\sin(x)$ and not $s \cdot i \cdot n \cdot (x)$, without any need for the dollar sign. The second extension to the grammar is the introduction of equations, which are nothing more than two expressions separated by an equals sign. The syntax for `2dplot` is a final extension, but only in the sense that the parser prepends the characters `$plot2d(` and appends a closing bracket, so that `$f(x), 0, 1` becomes `$plot2d($f(x), 0, 1)`. The function `plot2d` generates the graph when part of a `\2dplot/\end2dplot` construction.

There are several "system functions", and those which have been implemented are only a bare minimum, intended to allow one to write a few primitive scripts (i.e. the sample scripts in the Appendix). Which functions should actually be implemented is an area for further work. A serious problem which many of them have in common is the problem of zero recognition, which has been proven to be undecidable [25]. On the face of it, this makes it impossible to write a host of simply mathematical query functions. For example, is

$$x + \cos^2 y + \sin^2 y$$

a function of y in the sense that it varies when y is varied? This is an area which requires further work. What should be stressed at this stage is that we are interested in software which handles only a very restricted set of simple mathematical problems. In that context, undecidability proofs are not necessarily relevant. The software need only be able to solve problems a student can solve. The prototype's functionality is even more restricted.

The system functions implemented are

- `dangeroussimplifytimes0` applies the rules $x \cdot 0 \mapsto 0$ and $0 \cdot x \mapsto 0$ irrespective of what x is.

- `simplifytimes0` applies the rules $x \cdot 0 \mapsto 0$ and $0 \cdot x \mapsto 0$ if and only if x has a rational value (in a very strict sense – every subexpression must be a rational number).

- `simplifytimes1` applies the rules $x \cdot 1 \mapsto x$ and $1 \cdot x \mapsto x$.

- `simplifyplus0` applies the rules $x + 0 \mapsto x$ and $0 + x \mapsto x$.

- `expand` applies the rules

$$\begin{aligned} x \cdot (a+b) &\mapsto x \cdot a + x \cdot b, \\ (a+b) \cdot x &\mapsto a \cdot x + b \cdot x, \\ x \cdot (a-b) &\mapsto x \cdot a - x \cdot b \\ \text{and} \quad (a-b) \cdot x &\mapsto a \cdot x - b \cdot x. \end{aligned}$$

- `isinteger` returns true if every subexpression has an integer value. For example, `$isinteger(1+sin^2x+cos^2x)` returns false, but `$isinteger(6/2+7)` returns true.

- `isreal` returns true if its single argument has a finite floating point evaluation.

- `iszero` returns true if its single argument is zero (if it has an integer value which is zero, or, if a floating point expression, if its floating point evaluation is 0.0).

- `isgreaterthanzero` returns true if `isinteger` returns true and its integer value is greater than zero, or if its floating point evaluation is greater than zero.

- `islessthanzero` returns true if `isinteger` returns true and its integer value is less than zero, or if its floating point evaluation is less than zero.

- `isfunctionof` checks to see if the first argument is a function of only the others, using the naive assumption that the appearance of a variable in an expression makes that expression dependent on the variable. `$isafunctionof(x^2,x,y)` and `$isafunctionof(x^2+y,x,y)` return true. `$isafunctionof(x^2+y,x)` returns false. If there is only one argument, the function only returns true if no variable at all is present. For example, `$isafunctionof(2+x/7)` returns false.

- `identical` returns true only if both of its arguments are identical in every respect. `$identical(x^2+1+1,x^2+2)` returns false.

These are not only rather naive in the way they work, but (more importantly for this discussion) it is unclear whether these are functions which lead to "natural" scripts. It would seem that only experience in actually writing scripts can shed light on this question.

Note the fact that the function `expand` is careful not to change the order of multiplications (i.e. $x \cdot (a + b) \mapsto xa + xb$ but $(a + b) \cdot x \mapsto ax + bx$). We feel that this is a necessary feature, in line with our requirement that the automatic simplications made should not be more complex than those being discussed. Of course it would also be useful to have some sort of ordering function that does map ba to ab, but we feel that these two functionalities should be kept separate.

The most important commands are the selection commands `if`/`then`/`else`/`endif`. They control which text is in fact visible at any given time. The script

> The number of rabbits in the hat is
> **inputmaths** n=2 **endmaths**
> , or, putting this into words, we have
> **if** $iszero(n) **then**
>   no rabbits at all
> **else**
>   **if** $iszero(n-1) **then**
>   one rabbit
>   **else**
>   **if** $iszero(n-2) **then**
>   a pair of rabbits
>   **else**
>   many rabbits
>   **endif**
>   **endif**
> **endif** in the hat.

defines an interactive document which reads

> The number of rabbits in the hat is $n = 2$, or, putting this into words, we have
> a pair of rabbits in the hat.

initially, and then, depending upon what value the reader gives to n:

> The number of rabbits in the hat is $n = 0$, or, putting this into words, we have no rabbits at all in the hat.
>
> The number of rabbits in the hat is $n = 1$, or, putting this into words, we have one rabbit in the hat.
>
> The number of rabbits in the hat is $n = 1070032$, or, putting this into words, we have many rabbits in the hat.

4.2 The Prototype Equation Editor

The equation editor implements essentially all of the grammar presented in the Appendix (the equation editor was actually written before a grammar was defined). It implements automatic bracket pairing and automatic completion of expressions, and shows a cursor. It automatically inserts separating dots or spaces where these would help clarify which grouping is meant. Syntax errors result in the offending characters being displayed in red. A syntactically incorrect input will be automatically corrected when the reader presses RETURN/ENTER by deletion of the offending characters and automatic completion of the remainder of the input string. The only editing feature provided (and this makes the editor extremely primitive) is backspace, corresponding to deleting the last character of the string representing the displayed expression.

What is important about the equation editor in connection with the scripting language is that the equation editor accepts only characters which the grammar needs to define a mathematical expression (typing any other character has no effect). The dollar, which prefixes hidden function or variable names, is ignored. This is the mechanism by which hidden variables and "system" functions stay hidden. A reader is physically incapable of referring to them.

5 Conclusions

We have given a set of what we claim are necessary properties for an interactive mathematics textbook, defining in particular the distinct roles of the author and the reader. These requirements make certain demands on both the language in which such a textbook would be written (the scripting language), and the language in which the reader would interact with it. We have presented a prototype scripting language.

A linear grammar has been formally defined in an attempt to allow input and display notations to resemble each other as closely as possible. This grammar is based upon actual mathematical usage. For example, the expression

$$\sin^2\left\{2a\left|1-\frac{3}{7q}\right|\right\}\left[1+\frac{1}{2}\sin x\cos x\right]$$

is represented by the input string `sin^2{2a|1-3/7q|}[1+1/2 sinxcosx]`.

What is quite new about the grammar is its ability to cope with absolute value bars, so

$$\left|\frac{||-a|-b+|c||}{2}\right|$$

is represented by `|||-a|-b+|c||/2|` rather than the more cumbersome `abs(abs(abs(-a)-b+abs(c))/2)` which most computer algebra packages would demand. Such a linear grammar could become the basis of a general-purpose equation editor. An obvious extension of this work would be to implement such an equation editor as a portable software component, perhaps as a Java Bean [21] using OpenMath [26] to insure platform-independency.

The fact that most computer algebra packages are designed to solve large problems for experts means that they automatically perform simplifications which are more complex than those one might want to discuss in an elementary textbook, and are sometimes incapable of solving simple problems without extra information. What we actually need for interactive elementary mathematics textbooks is software designed to solve simple problems for non-experts.

The prototype demonstrates the concepts introduced here, especially concerning the relationship of a script written by the author to the actual document, and should also be seen as a proof of concept. Further work will concentrate on studying what functionality the computing engine actually needs to have for typical texts.

6 Acknowledgements

This work was partially funded by a grant from the Ingeniørenes Pædagogiske Nætværk (IPN) as part of the VIDIGEO project [27]. The author also wishes to thank Simon Kokkendorff, Peter Røgen, Prof. Steen Markvorsen and Guoxiao Yang for many useful discussions.

References

[1] M.B. Hayden and E.A. Lamagna, NEWTON: An Interactive Environment for Exploring Mathematics, *J. Symbolic Computation*, **25** (1998) 195–212.

[2] N. Kajler and N. Soiffer, A Survey of User Interfaces for Computer Algebra Systems, *J. Symbolic Computation*, **25** (1998), 127–159.

[3] `http://www.imath.org/iMathSystem/Introduction`

[4] `http://www.ma.utexas.edu/users/wfs/netmath/demo/demo.html`

[5] L. Lamport, *A Document Preparation System: LaTeX*, Second Edition, Addison-Wesley (1994).

[6] `http://xxx.lanl.gov/hypertex`

[7] `http://www.pragma-ade.nl`

[8] `http://www.cinderella.de`

[9] `http://integrals.wolfram.com`

[10] *Handbook of Mathematical Functions*, M. Abromowitz and I.A. Stegun (eds.), Dover Publications, New York (1970).

[11] N. Soiffer, Mathematical Typesetting in Mathematica, *ISSAC'95*, A.H.M. Levelt (ed.) (1995) 140–149.

[12] A. Heck, *Introduction to Maple*, Second Edition, Springer-Verlag, New York (1996).

[13] S. Wolfram, *The Mathematica Book*, Fourth Edition, Cambridge University Press, Cambridge (1999).

[14] D. Raggett and D. Batsalle, Adding math to Web pages with EzMath, *Computer Networks and ISDN Systems*, **30** (1998) 679–681.

[15] `http://www.w3.org/People/Raggett/EzMath/EzMathPaper.html`

[16] F. Cajori, *A History of Mathematical Notations*, Volumes I & II, The Open Court Publishing Company, La Salle, Illinois (1952).

[17] Y. Zhao, T. Sakurai, H. Sugiura and T. Torii, A Methodology of Parsing Mathematical Notation for Mathematical Computation, *ISSAC'96*, Y.N. Lakshman (ed.) (1996) 292–300.

[18] `http://www.webeq.com/webeq`

[19] A.V. Aho, R. Sethi and J.D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley Publishing Company, Reading Massachusetts (1986).

[20] M. Metcalf and J. Reid, *Fortran 90/95 explained*, Oxford Science Publications, Oxford, 1996.

[21] P. Niemeyer and J. Peck, *Exploring Java*, Second Edition, O'Reilly, Cambridge (1997).

[22] `http://www.w3.org/MarkUp/Activity`

[23] S.S. Dooley, Coordinating Mathematical Content and Presentation Markup in Interactive Mathematical Documents, *ISSAC'98*, O. Gloor (ed.) (1998) 54–61.

[24] `http://www.w3.org/Math`

[25] B.F. Caviness, On Canonical Forms and Simplification, *Journal of the ACM*, **17(2)** (1970) 385–396.

[26] S. Dalmas, M. Gaëtano and S. Watt, An OpenMath 1.0 Implementation, *ISSAC'97*, W.W. Kuchlin (ed.) (1997) 241–248.

[27] `http://www.mat.dtu.dk/VIDIGEO`

7 Appendix: Sample Script A

```
The \bold graph \plain of the function
\inputmaths
    P=sinx∧3/[2+cos2x]+x
\endmaths
\if $isfunctionof(P) \then
    is not very interesting because it is in fact just a constant.
\else
    \if $isfunctionof(P,x) \then
        for \literalmaths x \endmaths values between
        \inputmaths a=–3 \endmaths and
        \inputmaths b=–a+7 \endmaths
        \if $isreal(a) \then
            \if $isreal(b) \then
                \if $isgreaterthanzero(b–a) \then
                    looks like
                    \2dplot $P(x),a,b \end2dplot \skipline
                    You may zoom in on regions of the graph using the mouse,
                    and return to the original view by clicking once on it.
                    The function \literalmaths P \endmaths and the values
                    \literalmaths a \endmaths and \literalmaths b \endmaths
                    can be edited by clicking on them and then using
                    \italic backspace \plain and the other keys.
                \else
                    makes little sense because \showmaths a \endmaths
                    is greater than \showmaths b \endmaths .
                \endif
            \else
                is too complicated for me to draw because the
                value of \literalmaths b \endmaths
                that was just entered is not a number.
            \endif
        \else
            is too complicated for me to draw because the
            value of \literalmaths a \endmaths
            that was just entered is not a number.
        \endif
    \else
        is too complicated for me to draw because it is
        not just a function of \literalmaths x \endmaths .
    \endif
\endif
```

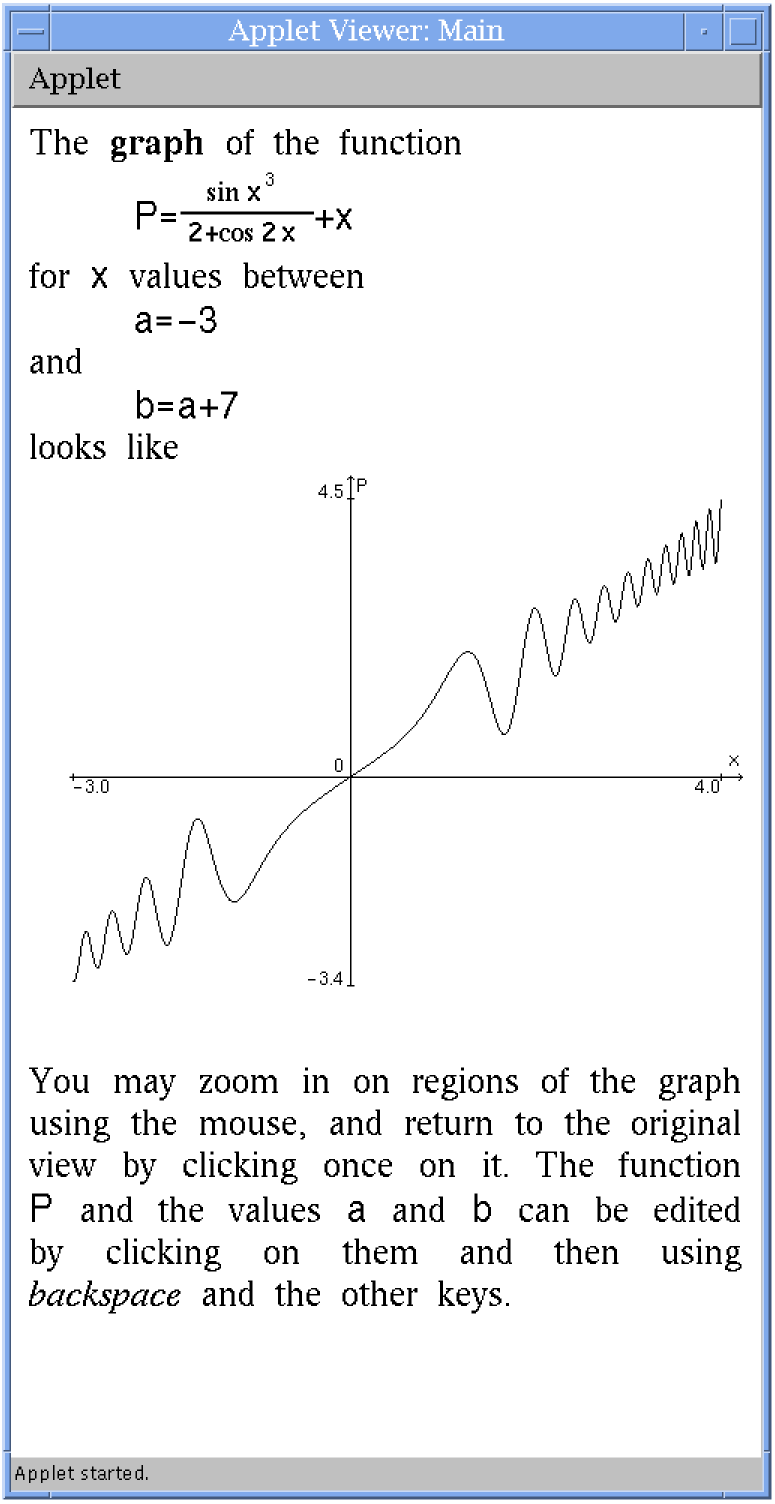
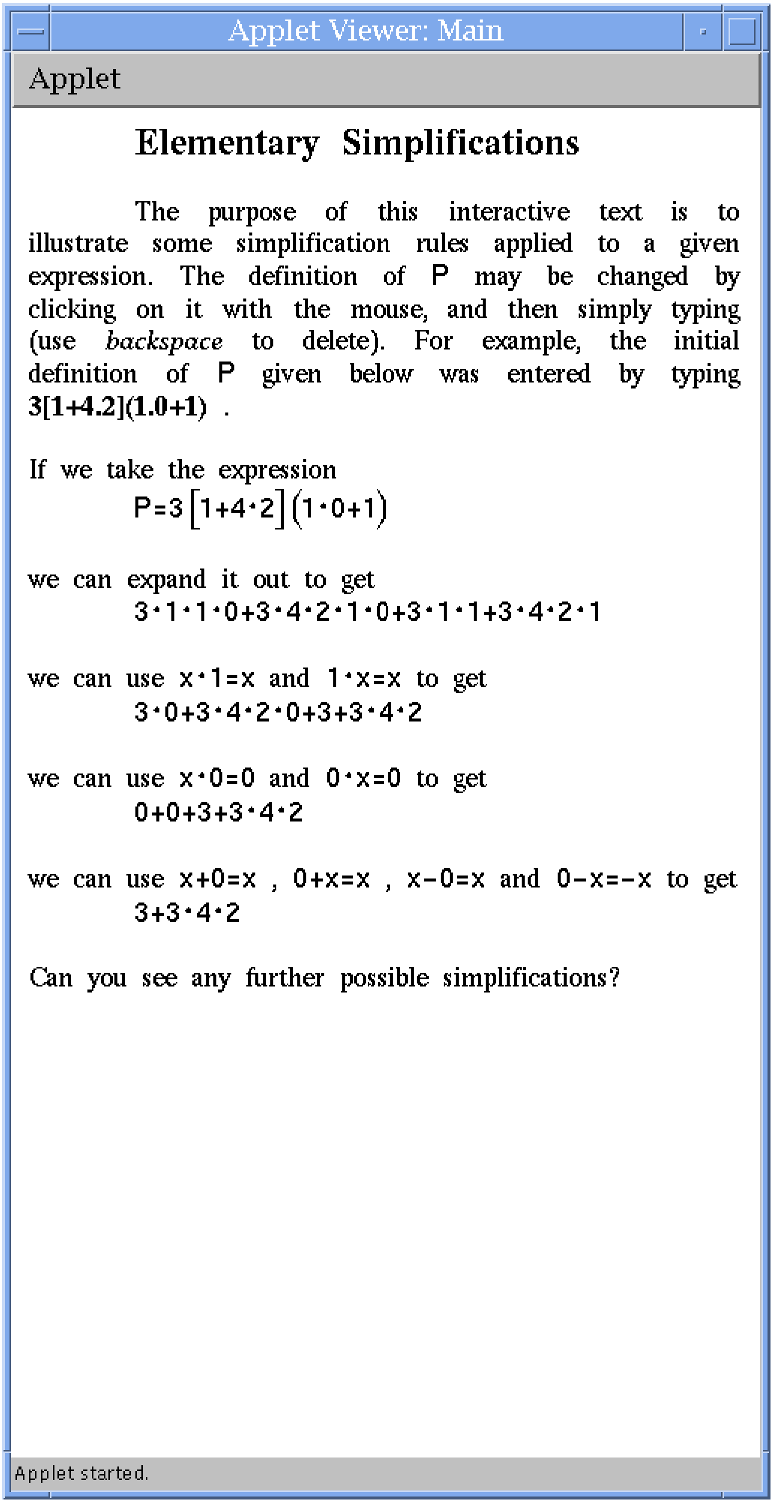


Figure 3: Sample script A: This is the display the reader sees first.

Figure 4: Sample script A: This is the display after the reader has changed the definition of P.

8 Appendix: Sample Script B

newparagraph **largesize** **bold** Elementary Simplifications **normalsize** **skipline** **newparagraph** The purpose of this interactive text is to illustrate some simplification rules applied to a given expression. The definition of **literalmaths** P **endmaths** may be changed by clicking on it with the mouse, and then simply typing (use **italic** backspace **plain** to delete). For example, the initial definition of **literalmaths** P **endmaths** given below was entered by typing **bold** 3[1+4.2](1.0+1) **plain** . **skipline**
If we take the expression **inputmaths** P=3[1+4.2](1.0+1) **endmaths**
hidemaths $zz=$expand(P) **endmaths**
if $identical($zz,P) **then** **else**
  skipline we can expand it out to get **newparagraph**
  showmaths $zz **endmaths** **endofline** **hidemaths** P=$zz **endmaths**
endif
hidemaths $zz=$simplifytimes1(P) **endmaths**
if $identical($zz,P) **then** **else**
  skipline we can use **literalmaths** x.1=x **endmaths** and
  literalmaths 1.x=x **endmaths** to get **newparagraph**
  showmaths $zz **endmaths** **endofline** **hidemaths** P=$zz **endmaths**
endif
hidemaths $zz=$simplifytimes0(P) **endmaths**
if $identical($zz,P) **then** **else**
  skipline we can use **literalmaths** x.0=0 **endmaths** and
  literalmaths 0.x=0 **endmaths** to get **newparagraph**
  showmaths $zz **endmaths** **endofline** **hidemaths** P=$zz **endmaths**
  hidemaths $ww=$dangeroussimplifytimes0(P) **endmaths**
  if $identical($zz,$ww) **then** **else**
    (but notice that **showmaths** $zz **endmaths** is not the same as
    showmaths $ww **endmaths** – we cannot use these identities if
    we are not sure of the finiteness of **literalmaths** x **endmaths**)
    endofline
  endif
endif
hidemaths $zz=$simplifyplus0(P) **endmaths**
if $identical($zz,P) **then** **else**
  skipline we can use
  literalmaths x+0=x **endmaths** , **literalmaths** 0+x=x **endmaths** ,
  literalmaths x–0=x **endmaths** and **literalmaths** 0–x=–x **endmaths**
  to get **newparagraph**
  showmaths $zz **endmaths** **endofline** **hidemaths** P=$zz **endmaths**
endif
skipline Can you see any further possible simplifications?



Applet Viewer: Main

Applet

Elementary Simplifications

The purpose of this interactive text is to illustrate some simplification rules applied to a given expression. The definition of P may be changed by clicking on it with the mouse, and then simply typing (use *backspace* to delete). For example, the initial definition of P given below was entered by typing **3[1+4.2](1.0+1)** .

If we take the expression

$$P=3\left[1+4\cdot 2\right]\left(1\cdot 0+1\right)$$

we can expand it out to get

$$3\cdot 1\cdot 1\cdot 0+3\cdot 4\cdot 2\cdot 1\cdot 0+3\cdot 1\cdot 1+3\cdot 4\cdot 2\cdot 1$$

we can use $x\cdot 1=x$ and $1\cdot x=x$ to get

$$3\cdot 0+3\cdot 4\cdot 2\cdot 0+3+3\cdot 4\cdot 2$$

we can use $x\cdot 0=0$ and $0\cdot x=0$ to get

$$0+0+3+3\cdot 4\cdot 2$$

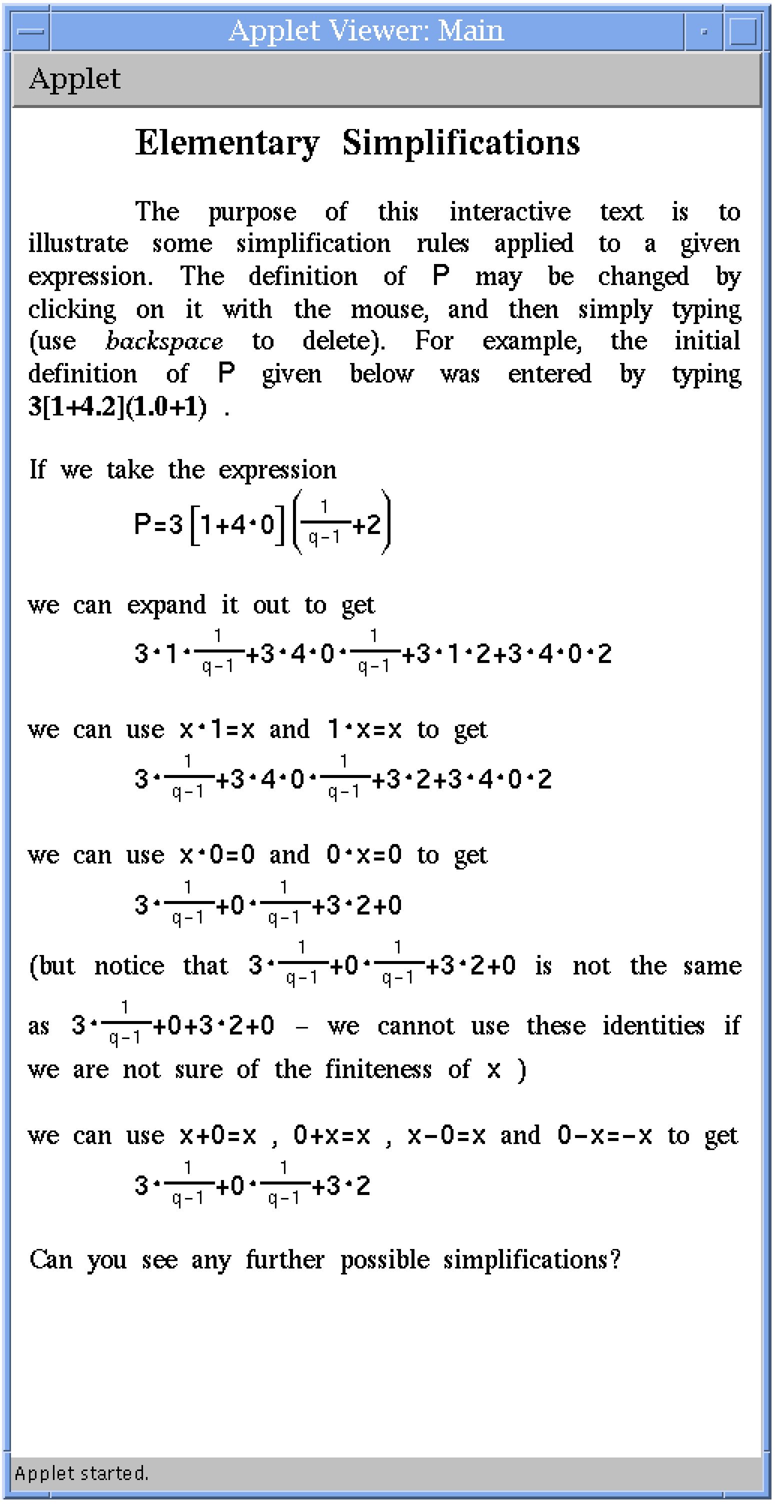
we can use $x+0=x$, $0+x=x$, $x-0=x$ and $0-x=-x$ to get

$$3+3\cdot 4\cdot 2$$

Can you see any further possible simplifications?

Applet started.

Figure 5: Sample script B: This is the display the reader sees first.



Applet Viewer: Main

Applet

Elementary Simplifications

The purpose of this interactive text is to illustrate some simplification rules applied to a given expression. The definition of P may be changed by clicking on it with the mouse, and then simply typing (use *backspace* to delete). For example, the initial definition of P given below was entered by typing **3[1+4.2](1.0+1)** .

If we take the expression

$$P=3\left[1+4\cdot 0\right]\left(\frac{1}{q-1}+2\right)$$

we can expand it out to get

$$3\cdot 1\cdot\frac{1}{q-1}+3\cdot 4\cdot 0\cdot\frac{1}{q-1}+3\cdot 1\cdot 2+3\cdot 4\cdot 0\cdot 2$$

we can use $x\cdot 1=x$ and $1\cdot x=x$ to get

$$3\cdot\frac{1}{q-1}+3\cdot 4\cdot 0\cdot\frac{1}{q-1}+3\cdot 2+3\cdot 4\cdot 0\cdot 2$$

we can use $x\cdot 0=0$ and $0\cdot x=0$ to get

$$3\cdot\frac{1}{q-1}+0\cdot\frac{1}{q-1}+3\cdot 2+0$$

(but notice that $3\cdot\frac{1}{q-1}+0\cdot\frac{1}{q-1}+3\cdot 2+0$ is not the same as $3\cdot\frac{1}{q-1}+0+3\cdot 2+0$ – we cannot use these identities if we are not sure of the finiteness of x)

we can use $x+0=x$, $0+x=x$, $x-0=x$ and $0-x=-x$ to get

$$3\cdot\frac{1}{q-1}+0\cdot\frac{1}{q-1}+3\cdot 2$$

Can you see any further possible simplifications?

Applet started.

Figure 6: Sample script B: After the reader has changed the definition of P. Note the warning against simplifying using the rule $x \cdot 0 \mapsto 0$ if it is not known if x is finite.

9 Appendix: Lex Specification

```
[ \t]+   return WS;
\+[ \t]* return PLUS;
\-[ \t]* return MINUS;
[0-9]+[ \t]+[0-9]+\/[0-9]+ {
  yylval.val.ipart=atoi(yytext);
  yylval.val.p=strrchr(yytext,' ')>strrchr(yytext,'\t')?
               atoi(strrchr(yytext,' ')+1):
               atoi(strrchr(yytext,'\t')+1);
  yylval.val.q=atoi(strchr(yytext,'/')+1);
  return JUX; }
[0-9]+\,[0-9]+ {
  yylval.val.ipart=atoi(yytext);
  yylval.val.fpart=atoi(strchr(yytext,',')+1);
  return DEC; }
[0-9]+   { yylval.val.ipart=atoi(yytext); return NUM; }
[a-zA-Z] { yylval.var=yytext[0];          return VAR; }
\\\/     return SQRT;
exp      return EXP;
sin      return SIN;
cos      return COS;
tan      return TAN;
sinh     return SINH;
cosh     return COSH;
tanh     return TANH;
ln       return LOG;
arcsin   return ASIN;
arccos   return ACOS;
arctan   return ATAN;
arcsinh  return ASINH;
arccosh  return ACOSH;
arctanh  return ATANH;
\n       return 0;
.        return yytext[0];
```

given the following union declaration

```
%union { int lab;
         struct rat { int ipart;
                      int fpart;
                      int p;
                      int q;     } val;
         char var; }
```

10 Appendix: Yacc Specification

The grammar presented here is ambiguous. There are 176 shift/reduce conflicts. These should all be resolved in favour of shift (Yacc does this automatically).

10.1 The "Outer" Grammar

<table>
<tr><td><code>exp_o:</code></td><td><code>sum_o</code></td><td></td></tr>
<tr><td><code>|</code></td><td><code>WS sum_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>sum_o:</code></td><td><code>sum_o PLUS prod_o</code></td><td>$\mathtt{sum\_o} + \mathtt{prod\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>sum_o MINUS prod_o</code></td><td>$\mathtt{sum\_o} - \mathtt{prod\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>PLUS prod_o</code></td><td>$+\mathtt{prod\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>MINUS prod_o</code></td><td>$-\mathtt{prod\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>prod_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>prod_o:</code></td><td><code>prod_o quot_o</code></td><td>$\mathtt{prod\_o} \cdot \mathtt{quot\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>prod_o '.' quot_o</code></td><td>$\mathtt{prod\_o} \cdot \mathtt{quot\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>prod_o '.' WS quot_o</code></td><td>$\mathtt{prod\_o} \cdot \mathtt{quot\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>prod_o WS</code></td><td></td></tr>
<tr><td><code>|</code></td><td><code>quot_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>quot_o:</code></td><td><code>pprod_o '/' pprod_o</code></td><td>$\frac{\mathtt{pprod\_o}}{\mathtt{pprod\_o}}$</td></tr>
<tr><td><code>|</code></td><td><code>pprod_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>pprod_o:</code></td><td><code>pprod_o '.' epow_o</code></td><td>$\mathtt{pprod\_o} \cdot \mathtt{epow\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>pprod_o epow_o</code></td><td>$\mathtt{pprod\_o} \cdot \mathtt{epow\_o}$</td></tr>
<tr><td><code>|</code></td><td><code>epow_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>epow_o:</code></td><td><code>fun_o</code></td><td></td></tr>
<tr><td><code>|</code></td><td><code>pow_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>pow_o:</code></td><td><code>atom_o '^' expon_o</code></td><td>$\mathtt{atom\_o}^{\mathtt{expon\_o}}$</td></tr>
<tr><td><code>|</code></td><td><code>atom_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>expon_o:</code></td><td><code>fun_o</code></td><td></td></tr>
<tr><td><code>|</code></td><td><code>atom_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
<tr><td><code>atom_o:</code></td><td><code>patom_o '!'</code></td><td>$\mathtt{patom\_o}!$</td></tr>
<tr><td><code>|</code></td><td><code>patom_o</code></td><td></td></tr>
<tr><td><code>;</code></td><td></td><td></td></tr>
</table>

`patom_o:`	`Patom`	
|	`bracabs`	
`;`		
`arg_o:`	`WS fpprod_o`	
|	`fun_o`	
|	`fpprod_o`	
|	`bracabs`	
`;`		
`fpprod_o:`	`fpprod_o fpow_o`	$\texttt{fpprod\_o}\cdot\texttt{fpow\_o}$
|	`fpprod_o bracabs`	$\texttt{fpprod\_o}\cdot\texttt{bracabs}$
|	`fpow_o`	
`;`		
`fpow_o:`	`Atom '^' expon_o`	$\texttt{Atom}^{\texttt{expon\_o}}$
|	`Atom`	
`;`		
`fun_o:`	`SQRT arg_o`	$\sqrt{\texttt{arg\_o}}$
|	`'\\' NUM '/' arg_o`	$\sqrt[\mathrm{NUM}]{\texttt{arg\_o}}$
|	`EXP arg_o`	$\exp(\texttt{arg\_o})$
|	`SIN arg_o`	$\sin(\texttt{arg\_o})$
|	`SIN '^' NUM arg_o`	$\sin^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`COS arg_o`	$\cos(\texttt{arg\_o})$
|	`COS '^' NUM arg_o`	$\cos^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`TAN arg_o`	$\tan(\texttt{arg\_o})$
|	`TAN '^' NUM arg_o`	$\tan^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`SINH arg_o`	$\sinh(\texttt{arg\_o})$
|	`SINH '^' NUM arg_o`	$\sinh^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`COSH arg_o`	$\cosh(\texttt{arg\_o})$
|	`COSH '^' NUM arg_o`	$\cosh^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`TANH arg_o`	$\tanh(\texttt{arg\_o})$
|	`TANH '^' NUM arg_o`	$\tanh^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`LOG arg_o`	$\ln(\texttt{arg\_o})$
|	`ASIN arg_o`	$\arcsin(\texttt{arg\_o})$
|	`ASIN '^' NUM arg_o`	$\arcsin^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`ACOS arg_o`	$\arccos(\texttt{arg\_o})$
|	`ACOS '^' NUM arg_o`	$\arccos^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`ATAN arg_o`	$\arctan(\texttt{arg\_o})$
|	`ATAN '^' NUM arg_o`	$\arctan^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`ASINH arg_o`	$\mathrm{arcsinh}(\texttt{arg\_o})$
|	`ASINH '^' NUM arg_o`	$\mathrm{arcsinh}^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`ACOSH arg_o`	$\mathrm{arccosh}(\texttt{arg\_o})$
|	`ACOSH '^' NUM arg_o`	$\mathrm{arccosh}^{\mathrm{NUM}}(\texttt{arg\_o})$
|	`ATANH arg_o`	$\mathrm{arctanh}(\texttt{arg\_o})$
|	`ATANH '^' NUM arg_o`	$\mathrm{arctanh}^{\mathrm{NUM}}(\texttt{arg\_o})$
`;`		

10.2 The "Inner" Grammar

```
exp:     sum
|        WS sum
;
sum:     sum PLUS prod      sum + prod
|        sum MINUS prod     sum − prod
|        PLUS prod          +prod
|        MINUS prod         −prod
|        prod
;
prod:    prod nquot         prod·nquot
|        prod '.' quot      prod·quot
|        prod '.' WS quot   prod·quot
|        prod WS
|        quot
;
nquot:   npprod '/' pprod   npprod / pprod
|        npprod
;
npprod:  npprod '.' fun     npprod·fun
|        npprod '.' powabs  npprod·powabs
|        npprod '.' pow     npprod·pow
|        npprod fun         npprod·fun
|        npprod pow         npprod·pow
|        fun
|        pow
;
quot:    pprod '/' pprod    pprod / pprod
|        pprod
;
pprod:   pprod '.' fun      pprod·fun
|        pprod '.' powabs   pprod·powabs
|        pprod '.' pow      pprod·pow
|        pprod fun          pprod·fun
|        pprod pow          pprod·pow
|        fun
|        powabs
|        pow
;
```

```
powabs:   atomabs '^' expon  atomabs^expon
|         atomabs
;
atomabs:  abs '!'            abs!
|         abs
;
abs:      '|' exp '|'        |exp|
;
expon:    fun
|         atomabs
|         atom
;
pow:      atom '^' expon     atom^expon
|         atom
;
atom:     patom '!'          patom!
|         patom
;
patom:    Patom
|         bracket
;
bracket:  '(' exp_o ')'      (exp_o)
|         '[' exp_o ']'      [exp_o]
|         '{' exp_o '}'      {exp_o}
;
arg:      WS fpprod
|         fun
|         fpprod
|         bracabs
;
bracabs:  bracket
|         abs
;
fpprod:   fpprod fpow        fpprod·fpow
|         fpprod bracket     fpprod·bracket
|         fpow
;
fpow:     Atom '^' expon     Atom^expon
|         Atom
;
Atom:     Patom '!'          Patom!
|         Patom
;
```

<table>
<tr><td>Patom:</td><td><code>NUM</code></td><td></td></tr>
<tr><td>|</td><td><code>DEC</code></td><td><code>ipart,fpart</code></td></tr>
<tr><td>|</td><td><code>JUX</code></td><td>$\mathtt{ipart}\frac{\mathtt{p}}{\mathtt{q}}$</td></tr>
<tr><td>|</td><td><code>VAR '_' NUM</code></td><td>$\mathrm{VAR}_{\mathrm{NUM}}$</td></tr>
<tr><td>|</td><td><code>VAR</code></td><td></td></tr>
<tr><td>;</td><td></td><td></td></tr>
<tr><td>fun:</td><td><code>SQRT arg</code></td><td>$\sqrt{\mathtt{arg}}$</td></tr>
<tr><td>|</td><td><code>'\\' NUM '/' arg</code></td><td>$\sqrt[\mathrm{NUM}]{\mathtt{arg}}$</td></tr>
<tr><td>|</td><td><code>EXP arg</code></td><td>$\exp(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>SIN arg</code></td><td>$\sin(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>SIN '^' NUM arg</code></td><td>$\sin^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>COS arg</code></td><td>$\cos(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>COS '^' NUM arg</code></td><td>$\cos^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>TAN arg</code></td><td>$\tan(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>TAN '^' NUM arg</code></td><td>$\tan^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>SINH arg</code></td><td>$\sinh(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>SINH '^' NUM arg</code></td><td>$\sinh^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>COSH arg</code></td><td>$\cosh(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>COSH '^' NUM arg</code></td><td>$\cosh^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>TANH arg</code></td><td>$\tanh(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>TANH '^' NUM arg</code></td><td>$\tanh^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>LOG arg</code></td><td>$\ln(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ASIN arg</code></td><td>$\arcsin(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ASIN '^' NUM arg</code></td><td>$\arcsin^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ACOS arg</code></td><td>$\arccos(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ACOS '^' NUM arg</code></td><td>$\arccos^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ATAN arg</code></td><td>$\arctan(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ATAN '^' NUM arg</code></td><td>$\arctan^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ASINH arg</code></td><td>$\mathrm{arcsinh}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ASINH '^' NUM arg</code></td><td>$\mathrm{arcsinh}^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ACOSH arg</code></td><td>$\mathrm{arccosh}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ACOSH '^' NUM arg</code></td><td>$\mathrm{arccosh}^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ATANH arg</code></td><td>$\mathrm{arctanh}(\mathtt{arg})$</td></tr>
<tr><td>|</td><td><code>ATANH '^' NUM arg</code></td><td>$\mathrm{arctanh}^{\mathrm{NUM}}(\mathtt{arg})$</td></tr>
<tr><td>;</td><td></td><td></td></tr>
</table>

11 Appendix: Sample Expressions

These examples are intended to illustrate the meaning of the grammar introduced in this paper by providing pairs of input strings and their mathematical content, as defined by the grammar. Recall that the input strings are never intended to be seen.

```
2cosa|b|c/7
```

$$\frac{2\cdot \cos\left(a\cdot |b| \cdot c\right)}{7}$$

```
2cosa.|b|c/7
```

$$\frac{2\cdot \cos\left(a\right) \cdot |b| \cdot c}{7}$$

```
2cosa |b|c/7
```

$$2\cdot \cos\left(a\right) \cdot \frac{|b| \cdot c}{7}$$

```
2cos a|b|c/7
```

$$\frac{2\cdot \cos\left(a\cdot |b| \cdot c\right)}{7}$$

```
2cos a.|b|c/7
```

$$\frac{2\cdot \cos\left(a\right) \cdot |b| \cdot c}{7}$$

```
2cos a |b|c/7
```

$$2\cdot \cos\left(a\right) \cdot \frac{|b| \cdot c}{7}$$

```
2cos(a|b|c)/7
```

$$\frac{2\cdot \cos\left(a\cdot |b| \cdot c\right)}{7}$$

```
2cos(a|b|c/7)
```

$$2\cdot \cos\left(\frac{a\cdot |b| \cdot c}{7}\right)$$

```
|2cosa|b|c/7|
```

$$|2\cdot \cos{(a)}| \cdot b \cdot \left|\frac{c}{7}\right|$$

```
|2cosa.|b|c/7|
```

$$\left|\frac{2\cdot \cos{(a)} \cdot |b| \cdot c}{7}\right|$$

```
|2cosa |b|c/7|
```

$$|2\cdot \cos{(a)}| \cdot b \cdot \left|\frac{c}{7}\right|$$

```
|2cos a|b|c/7|
```

$$|2\cdot \cos{(a)}| \cdot b \cdot \left|\frac{c}{7}\right|$$

```
|2cos a.|b|c/7|
```

$$\left|\frac{2\cdot \cos{(a)} \cdot |b| \cdot c}{7}\right|$$

```
|2cos a |b|c/7|
```

$$|2\cdot \cos{(a)}| \cdot b \cdot \left|\frac{c}{7}\right|$$

```
|2cos(a|b|c)/7|
```

$$\left|\frac{2\cdot \cos{(a\cdot |b| \cdot c)}}{7}\right|$$

```
|2cos(a|b|c/7)|
```

$$\left|2\cdot \cos{\left(\frac{a\cdot |b| \cdot c}{7}\right)}\right|$$

```
2xy_2z+15w-12/(q+z) [6^q-7] 4|W-3|
```

$$2{\cdot}x{\cdot}y_2{\cdot}z+15{\cdot}w-\frac{12}{(q+z)}{\cdot}\left[6^q-7\right]{\cdot}4{\cdot}\left|W-3\right|$$

```
1/2a+x^3 3.b+1/2 a^33(1-z)[1+1/z]-2a 1/3 c
```

$$\frac{1}{2{\cdot}a}+x^3{\cdot}3{\cdot}b+\frac{1}{2}{\cdot}a^{33}{\cdot}\left(1-z\right)\cdot\left[1+\frac{1}{z}\right]-2{\cdot}a{\cdot}\frac{1}{3}{\cdot}c$$

```
sin^2x+2sinxcosx+cos^2x/5+cos^2[x/5]
```

$$\sin^2\left(x\right)+2{\cdot}\sin\left(x\right)\cdot\cos\left(x\right)+\frac{\cos^2\left(x\right)}{5}+\cos^2\left[\frac{x}{5}\right]$$

```
3!/sinx|2 3+z|+coshx.cos hx+pcos(hx+p)
```

$$\frac{3!}{\sin\left(x{\cdot}\left|2{\cdot}3+z\right|\right)}+\cosh\left(x\right)\cdot\cos\left(h{\cdot}x\right)+p{\cdot}\cos\left(h{\cdot}x+p\right)$$

```
sin(x+1)y+siny(x+1)+17 2/a+95 2/3a
```

$$\sin\left(x+1\right){\cdot}y+\sin\left[y{\cdot}\left(x+1\right)\right]+17{\cdot}\frac{2}{a}+\left(95\frac{2}{3}\right){\cdot}a$$

```
-sin(1+n)x/3 + a|b|c|d^M/N| + a|b.|c|d^M/N|
```

$$-\frac{\sin\left(1+n\right){\cdot}x}{3}+a{\cdot}\left|b\right|{\cdot}c{\cdot}\left|\frac{d^M}{N}\right|+a{\cdot}\left|\frac{b{\cdot}\left|c\right|{\cdot}d^M}{N}\right|$$

```
sin(x)arcsin|-z|bc/2 + sin x arcsin|-z|b c/2
```

$$\frac{\sin\left(x\right)\cdot\arcsin\left|-z\right|{\cdot}b{\cdot}c}{2}+\sin\left(x\right)\cdot\arcsin\left|-z\right|{\cdot}b{\cdot}\frac{c}{2}$$

```
\/cos^7(5+3|2.|x|^[q/2]-1 3/4|)/{2-1/2!a b^3 z}
```

$$\frac{\sqrt{\cos^7\left(5+3{\cdot}\left|2{\cdot}\left|x\right|^{\left[\frac{q}{2}\right]}-1\frac{3}{4}\right|\right)}}{\left\{2-\frac{1}{2!{\cdot}a}{\cdot}b^3{\cdot}z\right\}}$$

ISSN 0904–7611

DEPARTMENT OF MATHEMATICS

DTU . BUILDING 303

DK-2800 LYNGBY