# Mathʃpad: A System for On-Line Preparation of Mathematical Documents

Roland Backhouse, Richard Verhoeven and Olaf Weber*

Department of Computing Science,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB Eindhoven,
The Netherlands.

{rolandb,mathpad}@win.tue.nl

September 26, 1996

**Abstract**

Mathʃpad is a structure editor designed to facilitate the writing of documents of a mathematical nature. The principal element of Mathʃpad is a system for defining and using "stencils" which define the visual and logical structure of notational elements in a document.

This paper overviews the main features of the editor from a user's viewpoint. Particular attention is given to a discussion of the priorities that guided the system's design — (on screen) readability, flexibility, writability and the integration of prose with mathematical calculation.

## 1 Introduction

> Mathematics is known in the trade as *difficult*, or *penalty*, *copy* because it is slower, more difficult, and more expensive to set in type than any other copy normally occurring in books and journals.

Scientists have an overwhelming desire to communicate the results of their efforts as effectively as possible and will do their utmost to ensure that the readability of their books

*Now at CWI, Amsterdam

and articles is the best that technology will permit. The quality of typesetting is a particularly vital issue for scientists whose work includes a large amount of mathematics since, as has long been recognised (see the quotation above, taken from [10]), the typesetting of mathematics is fraught with difficulty.

In the days that typesetting was done by professional printers, taking instructions from professional copy-editors, the authors' efforts could be concentrated on the content rather than on the appearance of their work. Nowadays, with computerised document preparation systems, the responsibility for the appearance of a document is falling more and more on the authors' shoulders. They are paying the "penalty" for mathematical copy, not in cash but in time and effort.

Computerised document preparation systems typically fall into two types: "visual design" and "logical design" systems. Visual design systems, often called "what-you-see-is-what-you-get" (WYSIWYG) systems, emphasise the final appearance of the document; logical design systems emphasise the logical structure.

Probably the best known example of a logical design system is the LaTeX system [8] developed by Lamport building on the TeX system designed by Knuth [7]. TeX and LaTeX have been described by Leonhard and Simon [9] as both "wonderful" and "awful" and their users as "masochists". They are "wonderful" because of their versatility and the superb quality of the document that they produce; it is probably for this reason (rather than the fact that the systems are freely and widely available, unlike most WYSIWYG systems) that many thousands of authors and a large number of publishers of scientific journals favour their use. They are truly "awful" because the input and editing of documents is based on teletype technology and makes no concessions to human beings. (The comparison we would make is with editing computer programs by splicing a paper tape — as one was obliged to do in the days before punch cards and keyboard input.)

Visual design systems, of which there are several commercial products available, are typically less painful and frustrating to use, but are less versatile and produce a lower quality document. Most importantly, they do not cater adequately for the specific needs of mathematical articles.

Mathʃpad is a system that attempts to combine the advantages of logical and visual design systems for the preparation of all kinds of documents, but in particular documents involving substantial amounts of mathematical material. Mathʃpad is a structure editor; that is, a Mathʃpad document is structured, not just a sequence of characters, and the primitive edit operations transform entire structures, not just single characters. The structure that is edited is defined by means of so-called "stencils" which provide the user with two "views" of the abstract document: an on-screen view and an "output" view. The on-screen view allows mathematics to be shown on screen in a form that is as close to conventional mathematical notation as technology permits whereas the output view is a file written in an ASCII-based markup language (most commonly, but not necessarily, a combination of TeX and LaTeX).

Mathʃpad aims to be more than just a document preparation system used for preparing articles or books for final publication. It aims to assist in the process of writing mathematics. Embodied in the design of the system is a focus on a small set of requirements that we

feel have been ignored for too long by designers of document preparation systems. In this paper we present, in turn, the *need* that Mathʃpad is intended to fulfil and the experience that led up to its identification, the *priorities* that we set in trying to fulfil that need, and the principal *features* of the Mathʃpad system that realise the chosen priorities. The paper is concluded by an assessment of what has been achieved.

# 2 The Need and the Priorities

The *need* that Mathʃpad attempts to fulfil is the ability to write at a computer mathematical documents, ultimately of publication quality, in which mathematics and text are completely integrated. In this section we discuss the history that led up to the identification of this as our most basic need. We discuss the work of the group that Mathʃpad was intended to support, our experience with a prototype of the Mathʃpad system and how that influenced our thinking, and the final conclusion of this preliminary endeavour.

## 2.1 The Calculational Method

Mathʃpad was developed to support the work of a research group active in the area of the mathematics of (computer) program construction. The nature of the group's work is such that their articles frequently involve substantial algebraic calculations in advanced areas of mathematics. The main tools used by the group to carry out their work were (and still are) paper and pencil.

The group's work is intended to advance the "calculational method". (See [2] and [5] for illustrations of the method.) The primary goal is to reduce as much as possible of program construction to straightforward syntactic calculation. The prototype of the Mathʃpad system, developed by Paul Chisholm [3], enabled the group to experiment with performing calculations directly at a computer terminal rather than doing them first on paper.

## 2.2 Avoiding the Introduction of Error

Performing algebraic calculations at a computer terminal and doing them by hand with paper and pencil are quite different sorts of activities. In designing Mathʃpad we started out from the viewpoint that the latter (using paper and pencil) is much more reliable than the former (using a computer). This is quite contrary to prevailing claims made about formal verification systems and the like. One must recognise, however, that, for us as human beings, paper and pencil is so much *handier*; the computer, in comparison, is a very clumsy instrument, and the use of a clumsy or badly-designed tool will inevitably introduce errors of its own kind. Avoiding the *introduction* of error was thus our first and foremost consideration.

The traditional argument for the use of a computer for algebraic calculations, that computers are able to faithfully execute formal manipulation rules, is one that we find

difficult to justify in the context of our own research. The problem is that it is frequently hard to explain to a dumb machine how a rule should be applied in a particular situation because there are —from the trained scientist's point of view— so many trivial intermediate steps that also have to be explained. A simple example serves to illustrate our point. Given functions $f$, $g$, and $\theta$ such that $\theta$ has inverse $\theta^{-1}$, we have the (straightforward) identity,

$$\theta \circ f = g \circ \theta \quad \equiv \quad f \circ \theta^{-1} = \theta^{-1} \circ g \quad .$$

(Here "$\circ$" denotes composition of functions.) Two examples of this identity are provided by the logarithm and exponential functions. The functions are inverses of each other, and, as is well known,

$$\mathsf{exp}(-x) = \frac{1}{\mathsf{exp}(x)} \qquad \text{whereas} \qquad -\mathsf{ln}(x) = \mathsf{ln}(\frac{1}{x}) \quad ,$$

and

$$\mathsf{exp}(x+y) = \mathsf{exp}(x) \cdot \mathsf{exp}(y) \qquad \text{whereas} \qquad \mathsf{ln}(x) + \mathsf{ln}(y) = \mathsf{ln}(x \cdot y) \quad .$$

In order to see the correspondence between the general property and the two specific instances one must recognise that the general property has been expressed in terms of functions alone, whilst the instances are in terms of functions applied to their arguments. One must also recognise that plus and times are both *binary* functions and that this must be taken into account in an explanation that the identity

$$\mathsf{exp}(x+y) = \mathsf{exp}(x) \cdot \mathsf{exp}(y) \quad \equiv \quad \mathsf{ln}(x) + \mathsf{ln}(y) = \mathsf{ln}(x \cdot y)$$

is indeed an instance of the general property. This can be difficult to explain to a fellow human being but it is nothing compared to the overhead that is typically required in order to command a computer program to apply the general rule.

The working scientist (or engineer) has a need to be able to get on with productive work and requires a tool that will allow him to communicate the justification for his calculation steps at the level of formality of his own choosing. The working scientist is very conscious of the possibility of error and will choose his tools carefully in order to avoid the danger of its introduction; the responsibility for *eliminating* error, however, remains his and is not delegated either to a machine or to a colleague. It is such scientists that Math∫pad aims to cater for.

## 2.3 Early Experience

The project leading to the construction of Chisholm's proof editor was formulated with the goal of constructing an on-screen "scratch pad" for doing symbolic algebraic calculations. Much emphasis was placed on designing a pleasant user interface with the hope that users would see the "scratch pad" as a viable replacement for paper and pencil. The resulting prototype system, the design of which was formulated in intensive discussions between Paul

Chisholm and Roland Backhouse, was implemented by Chisholm with technical support from Harm Paas, and became available for general use in 1990 [3].

Chisholm's proof editor was also a structure editor and its principal design element was a flexible mechanism for on-the-fly definition of mathematical notations (the "stencils" of the current system). In particular, the definition and use of associative operators was facilitated. In addition, a major element of the system was a transformation system allowing one to store and apply proof rules. LaTeX output was automatically generated and the emphasis was on the ergonomics. Finally, a number of features of the lambda calculus were built into the system. (The latter have never been used.)

During the two years from September, 1990, to September, 1992, our group developed a theory of datatypes based on relation algebra and used Chisholm's proof editor to expedite the documentation of the work. The development of this theory involved constructing a large number of original theorems; very little involved verifying known facts. There was never any compulsion on the members of the group to use Chisholm's editor; three members of the group (of five) used it to a reasonable extent, the others not at all.

In order to enhance the use of the proof editor, much effort was put into a careful organisation of a database of rules in relation algebra and the predicate calculus. The database contained several hundred rules split into several files. Some of the rules were included in more than one file in order to expedite searching for rules. That there were so many rules is not at all surprising. Nor is it surprising that all members of the group had fully mastered them all — good progress depended on our having done so.

In spite of all this preparation, the formal transformation system was hardly ever used! Most proofs were done "by hand" using the primitive structure editing facilities. We had great expectations of the transformation system but in the end we had to be honest with ourselves and admit that they had not been fulfilled.

In order to understand what went wrong, one has to have a good understanding of what is involved in constructing proofs of novel theorems.

Uninteresting (formal) proofs are proofs that consist solely of simplification steps. Interesting proofs involve at least one "complification" step: in an interesting proof one begins with an expression, makes it more complicated and then simplifies it. (The most interesting proofs repeat this process; it is common however for each repetition to be formulated individually as a lemma.)

A simplification step is, for example, replacing $x \times 1$ by $x$ (where $\times$ is some operator with unit 1); a complification step replaces $x$ by $x \times 1$. Complification is typical, for example, in proofs by induction. An excellent illustration is Henk Doornbos's relation-algebraic proof of Newman's lemma [4]. The first step in the proof is to replace the composition $R \circ S$ of two relations $R$ and $S$ by $R \circ I \circ S$ ($I$ being the unit of the composition operator). This is the hardest step of the proof (since one has to come up with the idea of introducing $I$ and deciding where in a larger expression it is to be placed); after that it is plain sailing — induction on the "$I$" just introduced.

Simplification steps are more obvious than complification steps. They are also easily carried out — using simple edit operations, no transformation system being needed provided that the one doing the calculation is fully conversant with the algebraic laws being

used. Complification steps are much less obvious because there are so many of them that may be applied at any one time. The use of a complification step is closely linked with the proof strategy, and assistance from a dumb computer is rarely of any assistance.

The possibility to organise rules into several files was included in Chisholm's proof editor in anticipation of the fact that a vast number of rules would be applicable at any stage in a proof. In requesting a list of applicable rules, one could specify a particular subexpression and a particular rule base. Even so, the mechanism never proved successful.

The transformation system built into Chisholm's editor was never useful in deciding on which rule to apply. Nor was it useful in applying simplification steps. But it was useful in applying complification steps. Take, for example, the star decomposition rule in regular algebra:

$$(a+b)^* = a^* \cdot (b \cdot a^*)^* \quad .$$

Note that the right side of this identity has a complicated form and that the variable $a$ is repeated. Such a rule (seen as a left-to-right rewrite rule) is difficult to apply using simple editing operations (particularly when the instantiations of the variables $a$ and $b$ are themselves complicated expressions). The possibility of introducing error is therefore great; the reliability of the digital computer in carrying out the rule is in this case a major asset.

Another reason for the failure of the transformation system was that the theory we were developing was unstable. New theorems were being continually developed and explored. As mentioned earlier, all members of the group had fully mastered the basic rules of relation algebra, predicate calculus etc. Most progress was made, however, by applying the newly invented theorems which, of course, were not part of the carefully organised database of rules. Attempts to keep the database up to date never got off the ground; the effort was not seen to be worthwhile.

Note that our assessment of this prototype editor may well have been different had we not been using the system to derive and document *novel* theorems. Moreover, one must bear in mind that the system was indeed a prototype with the attendant shortcomings. A transformation system modelled on Chisholm's may one day prove to be useful if one's goal is to teach the use of a particular algebra; we, ourselves, were experts in the algebras that we were using and had no need of a helping hand. A transformation system may also prove to be useful in circumstances where the formal system is very stable since then a great deal of effort can be expended in customising the transformation system. (Examples of such systems are the Maple and Mathematica systems.) Our own opinion is, however, that no formal system for program construction has yet reached a sufficiently high level of stability to merit the effort.

One of the major lessons we learnt from the prototype system was the importance of proportional spacing. Chisholm's editor enabled one to use a wide range of mathematical symbols, which was very welcome, but all symbols and the spacing between them had a fixed size. In using the editor we became very aware of how confusing monospaced text can be. It is very easy to illustrate the problem. Consider the distributivity of multiplication

over addition expressed using a typewriter font with a dot denoting multiplication:

```
(a+b).c = a.c+b.c
```

The intended structure of the right side is, of course, `(a.c)+(b.c)` but the use of a monospaced teletype font obliterates this structure. Because of the space around the dots, the eye tends to read the right side as three subexpressions `a`, `c+b`, and `c`. Compare this with how the expression would be automatically presented using the MathJpad system:

$$(a+b)\cdot c = a\cdot c + b\cdot c$$

The proportional spacing around the operators, and the operators' relative size, makes the structure of the formula immediately evident.

The reason that we became acutely aware of this seemingly trivial observation when using Chisholm's proof editor was the instanteneity with which transformations were executed by the system. The system could do things in a flash but the results still had to be read by the writer. Because the monospacing was often downright confusing, it was better to execute the transformation by hand in order to allow reading and transformation to keep pace with each other.

## 2.4 Integrating Mathematics and Prose

Right from the outset, beginning with the prototype system and extending through to the design of MathJpad itself, the development has been dominated by a basic philosophy towards mathematical work. Specifically, it is our belief that mathematics is not about mathematical "results". It is about the nature of the results: what the mathematics is about, why one should be interested in a particular calculation or theorem, and what one can learn from it. Doing mathematics is a literary exercise: a mathematical article is an essay, and a mathematics book is a novel complete with a plot and main characters. (We include in this writing computer programs which we also regard as a mathematical exercise.) We cannot envisage doing mathematics without being able to freely intersperse prose with mathematical calculations. Computerised math systems rarely cater for this viewpoint. It is often physically impossible to include explanatory text and where it is possible it is often the case that the inclusion of text is severely restricted.

## 2.5 The Priorities

Having identified our basic need, we now turn to the priorities we set in order to satisfy them.

The first priority was *readability*. Mathematical notation has been designed over centuries to be eminently readable, so a first requirement of the MathJpad system was that one should be able to work directly with mathematical notation (and not some approximation to mathematical notation).

Of course, we can all train ourselves to read the unreadable. Blind people learn to read braille and are able to cope admirably well. Those who have been programming for a quarter of a century or more will remember the time when it was necessary to be able to read paper tape in order to amend or correct a program. Reading paper tape is a skill that is soon learnt and, with practice, may appear to require little effort. The impediment it causes is, however, very substantial. Indeed, it is quite probably the case that the use of paper tape as input medium contributed far more in those days to the unreliability of computer software than did the lack of structured programming techniques.

Monospaced text, particularly when used for mathematical work, is also unreadable, although it has been around for so long that many ardent computer users do not seem to realise the training that has to be undergone to learn how to read it. We have already pointed out the confusion that monospaced text can cause. Matters are made quite intolerable when this is combined with the limitations of the ASCII character set (as happens in programming languages). It is indeed our contention that the continuing use of monospaced ASCII as primary input medium to computer programs is a significant handicap to program development. The belief that one can do creative mathematics in monospaced ASCII is as ridiculous as the belief that one can do creative mathematics in paper tape format.

Our second priority was *flexibility*. Mathʃpad was designed to support the work of a research group engaged in the exploration of novel mathematical theories and so flexibility was a must. But flexibility is also vital to all scientists. Even if one is —for example— designing an algorithm for some well-defined, specialist task it is likely that one needs to appeal to a whole host of existing mathematical theories. (It is often the case that it is only when one submits one's design to a completely formal proof that one becomes truly aware of just how many.) The use of a system designed to support only a subset of available mathematical knowledge will be more effort than it is worth.

Flexibility means that the responsibility for error lies fairly and squarely on the shoulders of the user. Formal proof assistants of which we are aware attempt to relieve the user of some of this responsibility by restricting the available possibilities. One often sees claims like: "the system guarantees that only type-correct statements can be entered". The idea of the "computer as protector" has however rightly been condemned [1] in a more general context. In the current context it is quite absurd. The discipline of thoroughness must remain an integral part of the scientific ethic and not be imposed by a machine. We have to throw away the idea of "expert systems" intended to replace the experts and get back to "systems for experts", i.e. systems designed to assist experts in a particular field in doing their work efficiently and effectively.

The third priority in the design of Mathʃpad was to support the *writing* of scientific articles rather than the production of a finished product. The papers that we ourselves write tend to undergo many revisions, often extending over years rather than months. The drafts we write need to be highly readable but often contain inadequate proofs — inadequate because they are too complicated, not at the right level of abstraction, incomplete, and sometimes downright wrong. The finished product is intended to show how the ideas should be formulated, and how the theorems should be proved, but it is rarely the case that the finished product represents how its results were first established. The time required

for the production of the finished product is also relatively short compared to the time required to reach a state in which a finished product is conceivable.

Verification systems seem to assume that their users are as inhuman as the system itself: always systematic, always tidy and well organised, always precise and always correct. (Or, if this is not the case, at least willing to be disciplined by the machine into such a mode of working.) They are intended to provide the final seal of approval on work that is already well developed and well organised. They do have a purpose but, in our view, a purpose that has been given an unduly high priority. A machine-supplied seal of approval has had no priority in the development of Mathʃpad.

### 2.6 What's in a name?

In summary, the need served by Mathʃpad is the ability to write about mathematics and mathematical calculations without the introduction of error that often occurs when transcribing handwritten calculations to computer readable form. Our priorities were readability of on-screen mathematics, flexibility with respect to the mathematical notations that could be employed, and a level of writability that would permit on-screen composition of mathematical texts.

The name "Mathʃpad" chosen for the system intends to capture these attributes: the integral sign emphasises the integration between the subject matter, "math", and the tool, a "pad" on which to write the mathematics. The integral sign can also be read as an "s" and then the logo is read as "maths-pad" emphasising the plurality of the mathematical systems that it is intended to support. Finally, the English word "pad" connotes something used for rough work rather than the finished product.

## 3 System Highlights

### 3.1 Stencils

The most important element in the design of Mathʃpad is formed by the so-called *stencils*. A Mathʃpad stencil is a file describing the notational conventions used in one or more Mathʃpad documents, and how these are to be set using an ASCII-based markup language (like LaTeX or HTML). Each stencil consists of a number of templates, each of which may have various versions.

In order to illustrate the use of stencils we have extracted various notational elements from the book "A Logical Approach to Discrete Math" by David Gries and Fred B. Schneider [5] and incorporated them in the stencil "Gries-Schneider". This stencil, as it appears on the screen is shown in fig. 1.

Like all authors, Gries and Schneider have their own style for presenting definitions, theorems etc., and their own notational conventions for constructing mathematical expressions. The Gries-Schneider stencil is in effect a syntax-directed editor for statements in their book. (This is, of course, only partially true. The stencil we have created contains

only a small number of templates in order to illustrate the general idea. Much more would be needed for the complete book. More importantly, not all the conventions used by Gries and Schneider are peculiar to their own book. For organisational reasons it is better to create several stencils, each forming a logical unit. For the examples below we have made use of other stencils, in particular a LaTeX stencil, in addition to the Gries-Schneider stencil.)

Figure 2 shows an extract from one page of their book (p.151); fig. 2(a) is a screen dump, and fig. 2(b) shows the extract after processing with LaTeX.

The Gries-Schneider stencil (fig. 1) illustrates the general form of a Mathʃpad stencil. There are six sections, labelled None, Prefix, Postfix , Associative, Left and Right. All but the first of these sections define mathematical operators and their precedences. The first section (the one labelled None) is used to define mathematical notations *and* other (not necessarily mathematical) conventions that are used in a document.

Ostensibly, seven conventional operators have been defined in the Gries-Schneider stencil — reading from top to bottom, $\neg$ , $\equiv$ , $\wedge$ , $=$ , . , $\Leftarrow$ and $\Rightarrow$ . The first is a prefix operator, the next three are all associative infix operators, the next two are left-associative infix operators, and the last one a right-associative operator. We say "ostensibly" because several of the operators have different "versions". If for instance one moves the mouse to point at "$\wedge$" in the stencil and presses down with the rightmost mouse button a small menu will appear containing two items "$\wedge$" and "$\vee$". The two operators have different meanings, of course, but according to the definition in the stencil, they are treated by Mathʃpad as different "versions" of the same operator. In fact the total list of operators defined in the stencil amounts to sixteen.

The postfix operator "$[\mathbf{V} := \mathbf{E}]$" is somewhat unconventional although its use should be evident from fig. 2. It illustrates a common phenomenon in mathematics, namely the use of an operator that is parameterised, in this case by a variable "**V**" and an expression "**E**". The operator "$\mathbf{O}\langle\mathbf{T}\rangle$" in the stencil is also parameterised by an operator "**O**" and text "**T**". It has been used to create the proof steps in fig. 2. Its use will be discussed further later.

Each operator has been assigned a precedence according to a list to be found at the beginning of Gries and Schneider's book. Different versions of an operator all have the same precedence. Although we don't discuss how here, it is very easy to add new operators to the stencil or to modify existing definitions. Doing so results in a simple syntax-directed editor for mathematical expressions. By clicking on entries in the stencil expressions can be entered into a Mathʃpad edit document. This can be done either top-down ("syntax-directed"), or from left to right, or indeed using a combination of top-down and left-to-right input.

The first entry in the None section of the Gries-Schneider stencil is labelled "rule". This template describes the style used by Gries and Schneider to present rules. Specifically, each rule typically has a label — "8.22" in the example, a name — "**Change of dummy**" in the example —, a condition or context under which the rule is valid — "Provided $\neg occurs(`y', `R, P')$ and $f$ has an inverse" in the example —, and the statement of the rule itself — "$(\star x \mid R : P) = (\star y \mid R[x := f.y] : P[x := f.y])$" in the example —.

Clicking on "rule" in the stencil with the left mouse button causes the following template to appear in your document at the location of the target selection:

(**T**) **Name:** **T**:
Condition: **T**,
Statement: **E**

The three bold **T**'s and the bold **E** are place holders for text and a mathematical expression, respectively. In Gries and Schneider's book the headings "**Name**", "Condition:" and "Statement:" do not appear. The template has been so defined that the headings *do* appear on the screen, in order to remind the writer of the function of each of the place holders, but *do not* appear in the LaTeX output generated by the Mathʃpad document. (Compare fig. 2(a) with fig. 2(b).) The Mathʃpad system incorporates an automatic mechanism for generating LaTeX output so that in principle it is unnecessary for the user to learn LaTeX in order to use the system. In practice it is sometimes desirable to override the system-generated LaTeX either, as in this case, because the screen version of a template is designed to assist in the *writing* of the document, or to improve the quality of the output. The general design principle that this illustrates is: the user is the boss and can always override automatic actions of the system.

The creation of (versions of) a template in a stencil can be very straightforward; it can also be the most difficult task of all in the use of Mathʃpad. It is very straightforward to enter simple mathematical operators. The difficulties begin when you require special layout conventions. The difficulties are compounded when the LaTeX automatically generated by Mathʃpad is not adequate to your needs. This is undoubtedly a major hurdle for the beginning user of Mathʃpad. The experienced user will, however, already have several stencils at their disposal and will seldom need to augment or modify them. When it is necessary to do so this can always be done "on-the-fly". Moreover, if a sufficiently large community of Mathʃpad users develops, we expect (hope!) that in time stencils for particular application areas (a C stencil, a category theory stencil, a calculus stencil etc.) will become readily available and will be shared among users thus making the initial hurdle for beginners much more palatable.

## 3.2 Expression Manipulation

The primary goal of Mathʃpad is to provide a pleasant, useable interface for performing algebraic calculations at a computer terminal. The argument for using a computer for such calculations rather than doing them by hand is a very simple one: computers are able to copy information very reliably and quickly whereas human beings tend to introduce mistakes in the process.

The fact that Mathʃpad is a structure editor pays enormous dividends in the ease with which substructures may be selected. To select an expression in a text it suffices to point the mouse at the principal operator in the expression and click. In other systems that we know of selection is effected by dragging the mouse — usually over some encoding of the

expression involving "funny" control characters or umpteen backslashes and macro names rather than the expression itself. The result is that the most basic editing operation, namely copying, becomes clumsy and hence unreliable in those systems. With Mathʃpad that is not the case.

Apart from being able to copy subexpressions easily and quickly, Mathʃpad provides simple-to-use functions for rearranging expressions, and for finding and replacing mathematical expressions of a given structure. The functions provided by Mathʃpad for expression manipulation are Copy and Swap, Reverse, Distribute and Factorise, Group and Ungroup, Apply , Rename and Find. The functions Swap, Copy and Find can also be used on plain text, but the remainder are peculiar to the manipulation of mathematical expressions.

Group and Ungroup are used to change the default structure of an expression. Apply is used to "apply" a function to an expression. (The operation is purely syntactic and not to be confused with evaluation of the function.) Rename is used to rename dummy variables within a given scope. We do not discuss these operations any further here.

Find is short for Find / Find-and-Replace. It can be used to search for, and possibly replace, both text and mathematical expressions. The effectiveness of Find is considerably enhanced by the fact that Mathʃpad is a structure editor. It can be used to search for expressions of a particular shape simply by specifying a pattern containing one or more unfilled place-holders. By combining this with a replacement pattern the operation can be used to transform expressions according to mathematical rules. For example, Find-and-Replace can be used to transform all (or some) expressions of the shape $(\mathbf{E1}{+}\mathbf{E2})^*$ to $\mathbf{E1}^* \cdot (\mathbf{E2} \cdot \mathbf{E1}^*)^*$ where **E1** and **E2** are place holders for expressions.

Infix operators play a vital rôle in mathematical formulae. This is due in no small measure to the relative importance of associativity among all the algebraic properties that an operator may enjoy. Because of the associativity of an operator like addition we can freely write, say, $a{+}b{+}c{+}d$ without fear of confusion. This is of great benefit to calculation because the notation is not biased to any particular grouping of the subexpressions. The exploitation of associativity in a calculation becomes an invisible step, one that we are often not consciously aware of.

A similar phenomenon lies at the heart of the use of infix notation for transitive relations. When we write, say, $a = b = c$ then we mean $a = b$ *and* $b = c$ . But, because of the transitivity of equality, it is also the case that $a = c$ . In an extended calculation in which several steps occur it is typically the equality between the first and last terms that we are interested in, the intermediate terms being only of passing interest. At the end of such a calculation it is nonetheless rare to see explicitly stated "and hence, *by transitivity of equality,* the first and last terms are equal". Such would only happen in a very elementary mathematical text, or in an extremely detailed (and pedantic) logical argument. The infix notation encourages us to *associate* the first and last terms with each other.

Several elements of Mathʃpad are geared to the use of infix notation in the two ways explained above. When creating a Mathʃpad stencil it is possible to declare that an operator is "associative". This does not mean that the operator is "associative" in the narrow mathematical sense exemplified by addition. It means instead that the operator can have a list of arguments, of arbitrary finite length, whereby the notation used to denote the

application of the operator to a list is to repeatedly write the operator between successive elements in the list of arguments. Thus $a+b+c$ denotes the application of the operator $+$ to the list $\{a\,,b\,,c\}$ . Equally, $a{=}b{=}c$ denotes the application of the operator $=$ to the list $\{a\,,b\,,c\}$ . These are both called "associative" in Mathʃpad jargon because the notation is intended to encourage one to "associate" terms in the expression together in a variety of different ways.

(It is also possible to declare an operator to be "left associative" or "right associative". Such a declaration has the conventional meaning. For example, if the operator $\Rightarrow$ has been declared to be right associative then the expression $a\Rightarrow b\Rightarrow c$ has the structure $a\Rightarrow(b\Rightarrow c)$ . )

The Mathʃpad functions that exploit the notion of associativity are Reverse, Distribute and its inverse Factorise. The function Reverse reverses the order of the elements in a list of arguments and (associative) infix operators. For example, reversing the expression $a\leq b<c{=}d\leq e$ results in the expression $e\leq d{=}c<b\leq a$ . (Note that the operation does not replace "$<$" by "$>$" or "$\leq$" by "$\geq$".) Subexpressions can also be reversed. Reversing the subexpression $(c\div d)\cdot e$ in $a\cdot b+(c\div d)\cdot e+f$ , for example, results in $a\cdot b+e\cdot(c\div d)+f$ . (Note that the operation is not recursive: the arguments of the division operator are not reversed.) Distribute and Factorise are most often used in the normal situation in a calculation where a function is known to distribute over a binary operator, for instance to take the step from $a\cdot(b{+}c{+}d)$ to $a\cdot b+a\cdot c+a\cdot d$ ("distribute") or vice-versa to go from $a\cdot b+a\cdot c+a\cdot d$ to $a\cdot(b{+}c{+}d)$ ("factorise").

The ease with which one can select, copy and rearrange expressions using these few simple operations makes Mathʃpad a very "handy" symbolic calculator, particularly in areas of mathematics where no specialised machine support is available. Yet more benefit can be obtained from the system if one is prepared to use it in seemingly unorthodox ways. The use of the template labelled "$\mathbf{O}\langle\mathbf{T}\rangle$" in the Gries-Schneider stencil is one such example. It defines a binary associative operator, just like the addition operator, except that the operator itself has two arguments: "**O**", which is used for the relation holding between the operator's two arguments in a proof step, and "**T**", which is used for the explanation of the proof step. (Note that the explanation can run over several lines: see fig. 2.) In this way no special "feature" has to be added to the system in order to accommodate the calculational proof style (or indeed other proof styles) used by Gries and Schneider in their book.

# 4 An Assessment

In this section we shall attempt an assessment of the Mathʃpad system. The assessment will necessarily be subjective.

Mathʃpad runs under the X window system on a Unix platform. The system is written in C and C++ and is freely available from `http://www.win.tue.nl/win/cs/wp/mathspad`. The distribution contains a number of stencils including a HTML stencil for writing World-Wide Web documents, a "latex2e" stencil with templates for most constructs in LaTeX2$_\varepsilon$

and a Z stencil for preparing Z specifications. All these are fully documented. There are a number of other stencils included in the distribution that are not documented. (The latex2e stencil was prepared by Hilary Backhouse and the Z stencil by Eerke Boiten.) An automatic installation script ensures that installation can be carried out painlessly without the assistance of "the local guru". The Mathʃpad manual is itself a Mathʃpad document and is distributed with the system. (The system includes a context-sensitive help facility which causes a part of the manual to appear on the screen. A similar facility is used to gain access to the documentation on stencils.) A PostScript version of the manual is also freely available so that a hard copy can easily be made.

Thanks to the World-Wide Web the system has now been downloaded from Eindhoven (the only site at which download statistics are available to us) several thousand times. We have also received approximately one thousand emails informing us of an attempt (mostly successful) to install the system. (The email is part of the automatic installation procedure, the installer being first asked for permission to send the message.) Finally, a handful of people have written to us about the system.

It is difficult to know how to assess these statistics. The number of downloads and installations indicates to us that the need for improved document preparation systems is felt by a great many scientists. As judged by the number of letters we have received requesting assistance, it is likely however that the system is actively used by only a very small number of scientists at this point in time. These are the only conclusions that we feel we can safely draw from the installation statistics.

The current system is robust and has been used to write a substantial number of documents including lecture notes, M.Sc. and Ph.D. theses, and (draft) books.

The learning curve needed to become proficient in the use of Mathʃpad is steep. Two reasons can be identified why this is so. The first is that too much knowledge of LaTeX is required of the user; the second is that Mathʃpad is a structure editor. Even so we are still convinced that it was right to develop what is primarily a structure editor for LaTeX documents.

The problem with LaTeX is that its use should be invisible, except when defining templates, but has been allowed to impinge on the design of the structure editor. This is most evident in the modal distinction that is made between expression and text placeholders — a bad design feature that is inherited directly from TeX's math and text modes.

The redeeming feature of Mathʃpad, that in our view justifies the extra effort of having to learn to use both LaTeX and the system itself, is its hybrid nature —WYSIWYG where it is needed but not a what-you-see-is-ALL-you-get system. In this respect we are at variance with other developers of mathematical tools. For example, Mika Seppälä, editor of the Euromath Bulletin, has written [13]

> TeXperts will immediately notice that Euromath does not offer the same quality of mathematical typesetting as TeX does. Euromath has, however, one important advantage: the editor is completely WYSIWYG and handles *structured* documents, offering powerful means for including semantics.

Where we disagree with Seppälä is in giving the quality of mathematical typesetting a

lower priority than the inclusion of "powerful means for including semantics". (The quoted passage contains an instance, albeit minor, of the difference in quality — "TeX" instead of "TEX".) The point is that the computer screen is a very primitive medium in comparison to the printed page, and is likely to remain so for a very long time to come. As we explained in the opening paragraphs of this paper, we believe that most scientists would place the quality of the final document as number one priority provided that this can be achieved without the introduction of error.

Structure editors in general have not achieved anywhere near the level of success that we had expected when we first set out to build such an editor in 1987. A possible explanation is that the designers of structure editors have fallen into the trap of the "computer as protector" paradigm. We have done our utmost to avoid this trap in the design of Mathʃpad and we believe that, given time, the extra burden of having to learn the principles of structure editing will become acceptable, at least for the user group of a system like Mathʃpad. Nevertheless we have to admit to having had disturbing experiences where attempts to explain structure editing in Mathʃpad to some of our colleagues have failed abysmally.

It is our firm belief that Mathʃpad offers a unique approach to the preparation of mathematical documents. There are, of course, many proprietary text editors that incorporate facilities for including mathematical equations. Typically these are not structure editors and are suitable only for small quantities of mathematics or are optimised for standard school (as opposed to advanced) level mathematics. Existing structure editors that we are aware of (for example [6, 11]) are "pre-compiled" editors based on the notion of an editor generator [12]. The disadvantage —to us all important— is their resulting inflexibility. The structures that one may edit must first be carefully defined and organised before generating the editor, at which point they become fixed and immutable. In contrast, Mathʃpad allows immediate on-line definition and redefinition of templates thus catering for rough, preparatory work.

In conclusion, the use of Mathʃpad is possibly not worthwhile if it is used in the way that systems like TEX and LATEX are traditionally used: that is for the preparation of the final version of a publication (especially when the task is delegated to a "TEXpert" and not done by the author). Mathʃpad does eliminate the drudgery of the many syntactic errors that can occur when using TEX or LATEX but entering mathematical text rather than keyboard characters does incur a penalty. It is our experience that the use of Mathʃpad truly pays dividends when it is used in the way for which it was intended — for composing mathematical texts. The use of a structure editor combined with the readability of WYSIWYG-like mathematics is the significant factor in avoiding error and in making revisions.

Contrary to our initial dreams, Mathʃpad has not eliminated our use of paper and pencil, which we continue to use for all but the simplest calculations. The primary fault lies not with Mathʃpad but with the computer. Computer screens are too small, their resolution is too low, and one is forced to adopt a rigid and unnatural upright posture. Computers remain clumsy instruments, to be recommended only for carrying out routine calculations. Creating mathematical documents remains difficult for which a penalty must still be paid. We believe, nonetheless, that Mathʃpad is a step in the right direction.

## Acknowledgement

Thanks go to the anonymous referee and to David Gries for their extremely helpful advice and detailed suggestions for improving this paper.

# References

[1] Apple Computer Inc. *Human Interface Guidelines: The Apple Desktop Interface.* Addison-Wesley Publishing Company, 1987.

[2] Roland Backhouse. The calculational method. In Roland Backhouse, editor, *Special Issue on The Calculational Method, Information Processing Letters*, volume 53, page 121. Elsevier, February 1995.

[3] P. Chisholm. Calculation by computer. In *Third International Workshop Software Engineering and its Applications*, pages 713–728, Toulouse, France, December 3–7 1990. EC2.

[4] H. Doornbos, R.C. Backhouse, and J. van der Woude. A calculation approach to mathematical induction. Technical report, Department of Computing Science, Eindhoven University of Technology, 1996. To appear in Theoretical Computer Science. Available via World-Wide Web at `http://www.win.tue.nl/win/cs/wp/papers`.

[5] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math.* Springer-Verlag, 1993.

[6] Ian Jacobs and Laurence Rideau-Gallot. A Centaur tutorial. Technical Report 140, INRIA Sophia-Antipolis, July 1992. Available at `http://www.inria.fr/croap/centaur/tutorial/main/main.html`.

[7] D.E. Knuth. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 1984.

[8] Leslie Lamport. *LATEX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1986.

[9] Woody Leonhard and Barry Simon. *PC Mom. The Mother of all PC Books.* Addison-Wesley, Reading, Massachusetts, 1994.

[10] Editorial Staff of the University of Chicago Press. *A Manual of Style.* The University of Chicago Press, Chicago and London, 1969.

[11] V. Quint and I. Vatton. Making structured documents active. *Electronic Publishing — Origination, Dissemination and Design*, 7(2):55–74, 1994.

[12] Thomas Reps and Tim Teitelbaum. *The Synthesizer Generator.* Springer-Verlag, New York–Heidelberg–Berlin, 1989.

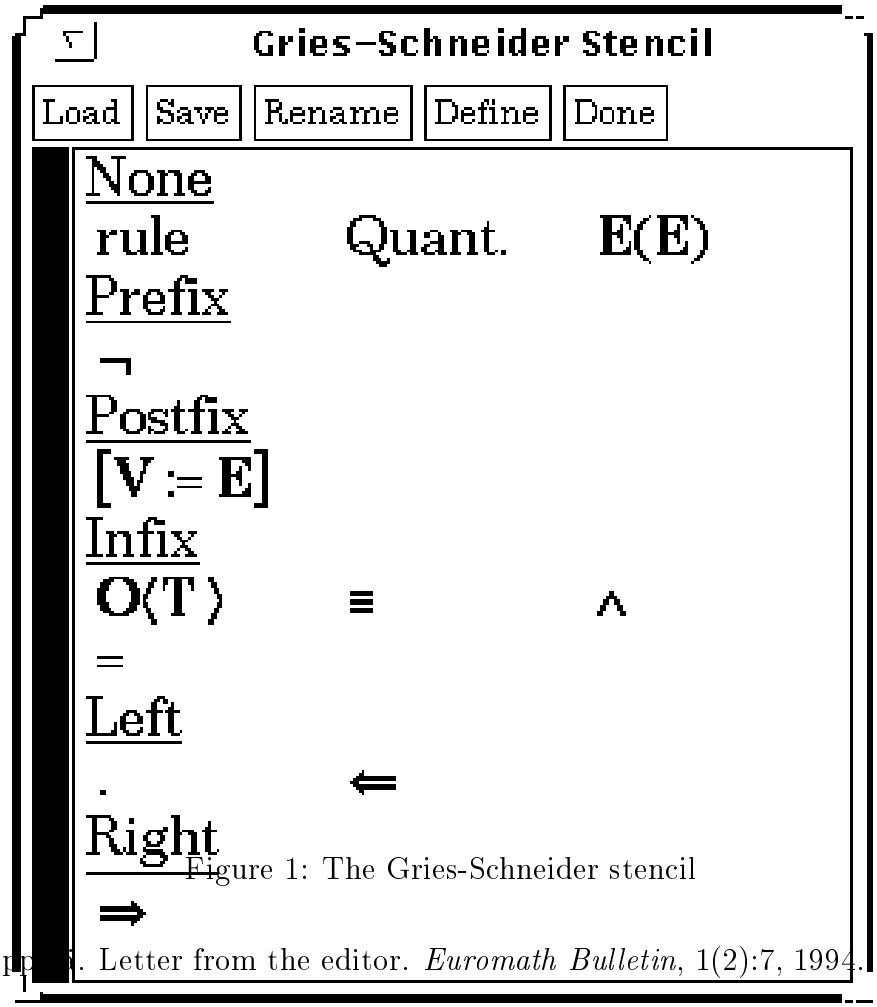
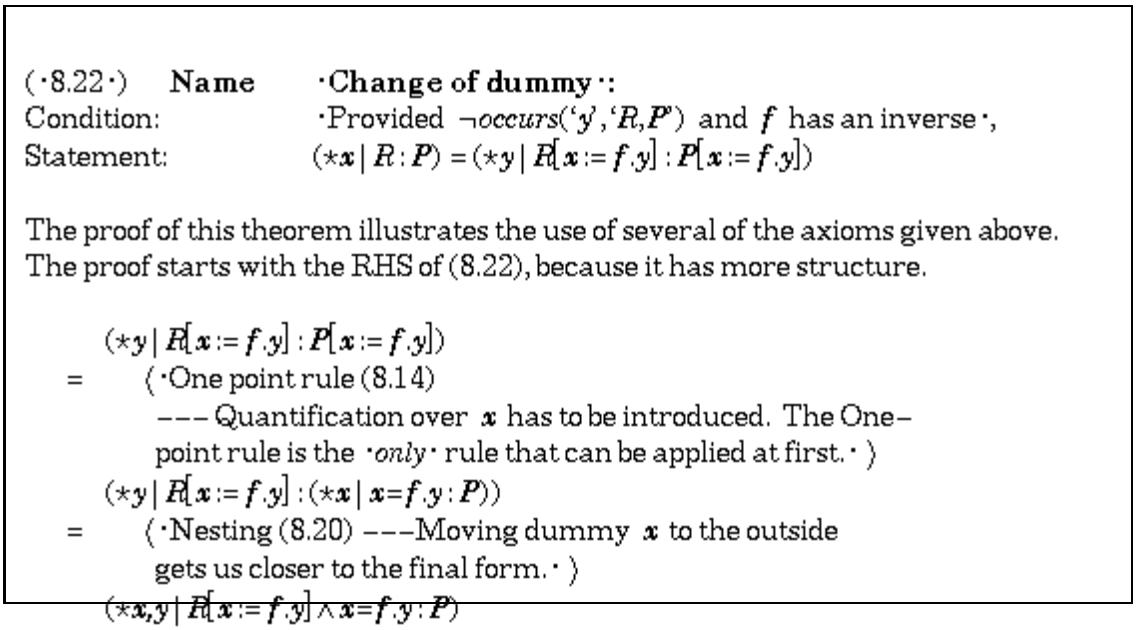



Figure 1: The Gries-Schneider stencil

[13] Mika Seppälä. Letter from the editor. *Euromath Bulletin*, 1(2):7, 1994.




**(a)** Snapshot of MathJpad document

(8.22) **Change of dummy**:
Provided $\neg occurs('y', 'R, P')$ and $f$ has an inverse,
$(\star x \mid R : P) = (\star y \mid R[x := f.y] : P[x := f.y])$
The proof of this theorem illustrates the use of several of the axioms given above. The proof starts with the RHS of (8.22), because it has more structure.

$(\star y \mid R[x := f.y] : P[x := f.y])$

$=$ ⟨One point rule (8.14)
— Quantification over $x$ has to be introduced. The One-
point rule is the *only* rule that can be applied at first.⟩

$(\star y \mid R[x := f.y] : (\star x \mid x = f.y : P))$

$=$ ⟨Nesting (8.20) —Moving dummy $x$ to the outside
gets us closer to the final form.⟩

$(\star x, y \mid R[x := f.y] \wedge x = f.y : P)$

**(b)** Output produced after processing with LaTeX

Figure 2: Running example. (Note the integration of prose with mathematical calculations)