Math∫pad
Tool Support for the Calculational Method

Roland Backhouse and Richard Verhoeven

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

{rolandb,mathpad}@win.tue.nl

19th February 1996

Abstract

Math∫pad is a structure editor designed to facilitate the integration of doing mathematics and writing about it. The principal element of Math∫pad is a system for defining and using "stencils" which define the visual and logical structure of notational elements in a document. Structure-editing features enable the use of Math∫pad to perform symbolic calculations, in non-standard algebras, directly on-screen.

This paper discusses the considerations that influenced the design of Math∫pad. Particular attention is given to a discussion of the priorities that guided the system's design. A brief review of the stencil mechanism is presented.

Engineering design is about creating artefacts that fulfil a need to the best of the ability of current technology. Good design involves insightful decisions on what is important, and what should be given priority. In this paper we present a personal assessment of the needs of those involved in the development of the mathematics of program construction. We discuss the nature of proof, and the priorities for software tools. Our view of such tools has been realised in Math∫pad, a system that facilitates the writing of mathematical documents. We briefly describe the main elements of the system and how they contribute to the achievement of our goals. We also briefly discuss future developments of the system.

1 The Need

The *need* that Math∫pad attempts to fulfil is the ability to write mathematical documents, of publication quality, in which mathematics and text are completely integrated[1]. 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 Math∫pad was intended to support, our experience with a prototype of the Math∫pad system and how that influenced our thinking, and the final conclusion of this preliminary endeavour.

1.1 The Calculational Method

Math∫pad 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 [4] for illustrations of the method.) The primary goal is to reduce as much as possible of program construction to straightforward syntactic calculation. The main obstacle to progress is the tension between concision and precision — present in all scientific disciplines but particularly acute in computer programming.

In order to achieve this goal the group neither shuns nor favours any one design method or calculus. In recent years, for example, the group has made contributions to constructive type theory, relation algebra and category theory (all in the context of program construction). The prototype of the Math∫pad 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.

1.2 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 Math∫pad 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.

[1] This explains the design of the trademark "Math∫pad": an 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.

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. The icon that is used for Math∫pad illustrates this point[2]. Given functions f, g, θ_1 and θ_2 such that θ_1 and θ_2 have inverses θ_1^{-1} and θ_2^{-1}, respectively, we have the (straightforward) identity,

$$\theta_1 \circ f = g \circ \theta_2 \quad \equiv \quad f \circ \theta_2^{-1} = \theta_1^{-1} \circ g \quad .$$

(Here "$\circ$" denotes composition of functions. The Math∫pad icon is slightly less general, since the subscripts on θ_1 and θ_2 are omitted.) 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.

[2] The example presented is actually an instance of a more general theorem. The chosen instance should be understandable to all readers. The more general theorem concerns adjunctions in category theory.

1.3 Early Experience

The project leading to the construction of the prototype 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 lattice theory. The database contained several hundred rules split into fifteen 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. At no time did we rely on the editor to remind us of any rule.

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 [1]. 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. Searching for and choosing a rule was too time consuming; hand editing was more effective.

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, lattice theory *etc.* Most progress was made, however, by applying the newly invented theorems which 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.

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 (particular 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.

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 optimising its useability. (Examples of such systems are the Maple and Mathematica systems.) Our own opinion is, however, that no formal system for program construction that 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 Math∫pad 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.

1.4 Integrating Mathematics and Prose

Right from the outset, beginning with the prototype system and extending through to the design of Math∫pad itself, the development has been dominated by a basic philosophy towards mathematical work. Specifically, it is our belief that mathematics is not just about mathematical calculations. It is about the nature of the calculations: 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.

This then was our need: to be able to write mathematical documents in which we could present and discuss mathematical calculations in non-standard algebras. In order to avoid the introduction of error caused by transcribing paper and pencil calculations into computer-readable form we needed a system that would allow us to conduct the calculations directly on screen. Finally, the system had to be pleasant and easy to use so that the act of using it would not interfere with creative thought.

2 The Priorities

Having identified our basic needs, 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 Math∫pad 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. It is a skill that is soon learnt and, after some practice, it is possible to convince oneself that it can be done without thinking. To do so is however pure self-deception. 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 the most significant single source of programming error. The belief that one can do creative mathematics in ascii is as ridiculous as the belief that one can do creative mathematics in paper tape format.

Our second priority was *flexibility*. Math∫pad 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. All 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".

Our view is that this is a totally misguided policy. 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 Math∫pad 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. When the finished product is finally ready it is usually the case that we feel a sense of shame at our own stupidity during the course of its production. 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 seems 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 Math∫pad.

3 System Highlights

3.1 Stencils

The most important element in the design of Math∫pad is formed by the so-called *stencils*. A Math∫pad stencil is a file describing the notational conventions used in one or more Math∫pad documents, and how these are to be set using an ascii-based mark-up language. Each stencil consists of a number of *templates*, each of which may have various *versions*. Each version defines two *views* of an abstract syntactic structure (not necessarily mathematical), the *screen* view and an *output* view. The screen view defines the appearance of the structure on the screen, the output view is the encoding of the structure in the chosen mark-up language (most commonly, but not necessarily, TeX or LaTeX [6, 7]).

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 [4]. 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,

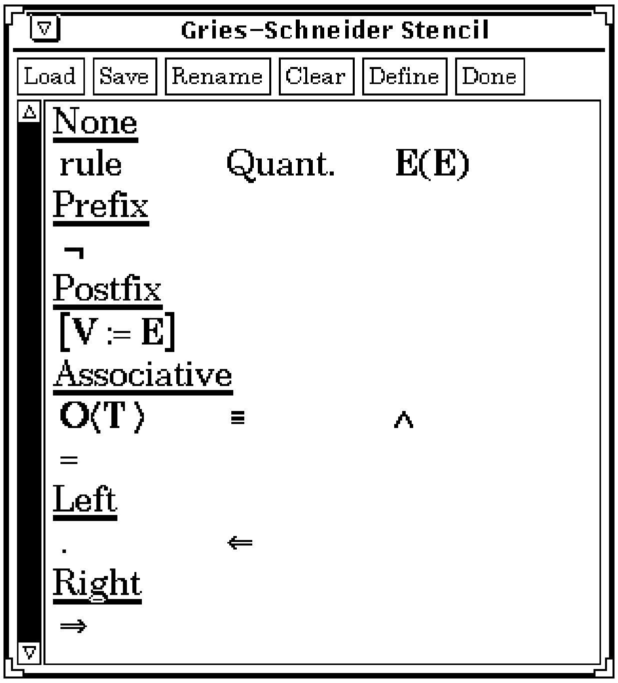

Figure 1: The Gries-Schneider stencil

theorems etc., and their own notational conventions for constructing mathematical expressions. The Gries-Schneider stencil is in effect a syntax-directed editor for (a subset of) statements in their book.

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 Math∫pad 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 four 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 Math∫pad as different "versions" of the same operator. In fact the total list of operators defined in the stencil amounts to sixteen.

The operator "[$\mathbf{V} := \mathbf{E}$]" is somewhat unconventional although its use should be evident

(·8.22·) **Name** **·Change of dummy·:**
Condition: ·Provided $\neg occurs(`y', `R,P')$ and f has an inverse·,
Statement: $(\star \boldsymbol{x} \mid R : P) = (\star y \mid R[\boldsymbol{x} := f.y] : P[\boldsymbol{x} := f.y])$

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.

$(\star y \mid R[\boldsymbol{x} := f.y] : P[\boldsymbol{x} := f.y])$
= ⟨·One point rule (8.14)
--- Quantification over $\boldsymbol{x}$ has to be introduced. The One–
point rule is the ·*only*· rule that can be applied at first.·⟩
$(\star y \mid R[\boldsymbol{x} := f.y] : (\star \boldsymbol{x} \mid \boldsymbol{x} = f.y : P))$
= ⟨·Nesting (8.20) ---Moving dummy $\boldsymbol{x}$ to the outside
gets us closer to the final form.·⟩
$(\star \boldsymbol{x}, y \mid R[\boldsymbol{x} := f.y] \wedge \boldsymbol{x} = f.y : P)$

(a) Snapshot of Math∫pad 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)

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 Math∫pad 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 Math∫pad document. (Compare fig. 2(a) with fig. 2(b).) The Math∫pad 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 over-ride 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 over-ride automatic actions of the system.

3.2 Expression Manipulation

The primary innovation of Math∫pad 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 Mathpad 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. This is particularly vital to avoiding the introduction of error. Computers are able to copy with perfect precision. Errors are introduced, nevertheless, when the wrong selection is made. And that is only too easy to do with unstructured textual representations of mathematics.

Apart from being able to copy subexpressions easily and quickly, Mathpad provides simple-to-use functions for rearranging expressions, and for finding and replacing mathematical expressions of a given structure. The functions provided by Mathpad 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.

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 Mathpad 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 $\mathbf{E1}^*\circ(\mathbf{E2}\circ\mathbf{E1}^*)^*$ 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 Mathpad are geared to the use of infix notation in the two ways explained above. When creating a Math∫pad 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 Math∫pad jargon because the notation is intended to encourage one to "associate" terms in the expression together in a variety of different ways.

The Math∫pad 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 Math∫pad 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 Conclusion

Ultimately, the success or otherwise of a system like Math∫pad must be judged by its users. That Math∫pad fulfills a need shared by many seems to be borne out by the enthusiasm with which it has been downloaded. (The system, which runs under the X window system, is freely available at the URL `http://www.win.tue.nl/win/cs/wp/mathspad`.) The first official announcement of the system's existence occurred in late October, 1995, via the comp.os.linux.announce newsgroup. Although this is the only place that we have actively advertised the system, by the end of December, 1995, we had received 270 email messages reporting that the system had been successfully installed. [3]

The fact that many have taken the effort to install the system does not say anything about the proportion that have found it to their liking. In this respect, we have much less information. The reaction we have had has been exclusively enthusiastic, one correspondent referring to Math∫pad as a "wonderful tool" that had doubled his productivity. Of course, this does not mean that the system is perfect, but we hope that it makes a useful contribution to current technology.

Now that we have a pleasant and reliable editor, we are exploring interfacing it to other systems. Our goal is to turn the system into a "literate programming" environment [5]in which programs can be written in an application-specific syntax. In particular, we want to remove the divide that exists between formal specifications of computer software —typically written in a mathematical language— and their implementations —written in a conventional, ascii-based, programming language. We believe that Math∫pad's stencils provide the appropriate mechanism to do this, by the simple device of simultaneously allowing different views on the same abstract structure. We also believe that Math∫pad's emphasis on readability, flexibility and writability as the first priorities give us an excellent starting point from which to begin.

References

[1] R.C. Backhouse and H. Doornbos. Mathematical induction made calculational. Technical report, Department of Computing Science, Eindhoven University of Technology., 1994. Available via world-wide web at `http://www.win.tue.nl/win/cs/wp/papers`.

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

[5] D.E. Knuth. Literate programming. *Computer Journal*, 27(2):97–111, 1984.

[6] Donald E. Knuth. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 1992.

[3] Built into the installation routine is a mechanism that sends us an email (subject to the consent of the installer) of the success or otherwise of the installation procedure. The total number of successful installations could be much higher since the system may have been installed by many individuals without direct email connection. About 100 installations were unsuccessful, primarily due to missing systems software.

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