Design Principles of *Mathpert*: Software to Support Education in Algebra and Calculus

Michael Beeson
Department of Mathematics and Computer Science
San Jose State University
San Jose, California 95192
USA
email: beeson@cats.ucsc.edu

February 14, 1996

This paper lists eight design criteria that must be met if we are to provide successful computer support for education in algebra, trig, and calculus. It also describes *Mathpert*, a piece of software that was built with these criteria in mind. The description given here is intended for designers of other software, for designers of new teaching materials and curricula utilizing mathematical software, and for professors interested in using such software. The design principles in question involve both the user interface and the internal operation of the software. For example, three important principles are *cognitive fidelity*, the *glass box* principle, and the *correctness* principle. After an overview of design principles, we discuss the design of *Mathpert* in the light of these principles, showing how the main lines of the design were determined by these principles.[1]

1 Purposes of software for mathematics education

The first step in proper software design is a clear statement of the purpose of the software. Here, for example, is a statement of purpose for *Mathpert*:

Mathpert is intended to replace paper-and-pencil homework in algebra, trig, and calculus, retaining compatibility with the existing curriculum while at the same time supporting innovative curriculum changes; to provide easy-to-use

[1] The scope of this paper is strictly limited to an exposition of the design principles and their application to *Mathpert*. I shall not attempt to review projects other than *Mathpert* in the light of these design principles.

computer graphics for classroom demonstrations in those subjects, as well as for home study; to replace or supplement chalk-and-blackboard in the classroom for symbolic problems as well as graphs. *Mathpert* is not intended to replace teachers or books: it is not explicitly tutorial. It is a "computerized environment for solving problems." It can be used by students of all ages and levels who are prepared to learn the subject matter.

Most experiments to date with using software in calculus instruction have used general-purpose symbolic programs such as *Maple* or *Mathematica*. These programs were not developed specifically for education, and it is therefore small wonder that their capabilities are not entirely matched to the needs of students.

I will show in this paper, among other things, that if we start with an *educational* purpose, and enunciate some simple design principles that more or less obviously follow from that purpose, these principles have ramifications that run through to the computational core of such a system, so that it is impossible to achieve ideal results by tacking on some additional "interface" features to a previously existing computation system. To put the matter another way: it is not possible entirely to separate "interface" considerations from "kernel" considerations. Such a separation might be possible (to facilitate running on different platforms), but only if the kernel itself has been designed with education in mind.

The most serious and thorough attempt to build an educational system based on pre-existing symbolic computation software is described in [17]. The first paragraph of this paper says "we will demonstrate how specific features common to symbolic computation programs diminish their pedagogical effectiveness." Nevertheless, they say "The reason to adopt an existing package rather than develop ones own is the recognition that programs like Maple and Mathematica represent massive programming efforts that do not need to be repeated." They identify many of the same problems with computation systems that are discussed in this paper. In order to overcome the problem of incorrect inferences, they developed a separate "derivation system" of "semantic machinery", which runs between Maple and their user interface. Also, a good deal of programming in the Maple language was necessary to produce student-sized steps, rather than mystifying answers. A more detailed comparision of their system with *Mathpert* is beyond the scope of this paper. Their work does not constitute a counterexample to the claim in the text, but rather additional support: rather than "tacking on" an interface to Maple, it was found necessary to perform many person-years of system development in both C and Maple to obtain a usable system.

Although *Mathpert* is not designed to replace teachers and books, there is a demand for stand-alone software that would allow a student to work alone, independent of teachers and books. Therefore *Mathpert* has been designed in such a way that it can later be incorporated in or used by systems which *are* explicitly tutorial. This point will be taken up in the last section of this paper.

2 Design principles

In this section I present and discuss eight fundamental design principles that guided the design of *Mathpert*.

2.1 Cognitive fidelity

We call mathematical software *cognitively faithful* if it satisfies the following criterion. When generating solutions, as opposed to supporting student-generated solutions, the software solves problems as the student should. This means that it takes the same steps as the student should take, in a correct order. Of course, there may be several acceptable "solution paths", all of which will be accepted if the student takes them; but the one which the computer generates if requested to do so must be exemplary in every respect. In particular, high-powered algorithms based on advanced methods are problematic. Most modern computer algebra systems use such advanced methods for fundamental operations such as factoring polynomials, solving equations, and computing integrals. What is a student to think when she asks for a factorization of $x^5 + x + 1$ and receives the answer $(x^2 + x + 1)(x^3 - x^2 + 1)$?

She will have no idea how such a factorization can be obtained, even though the result can be verified by multiplying out. For a student who has mastered elementary algebra, it might be quite a valuable aid to genuine mathematical exploration to have such an "oracle" available. But, the computer needs to present elementary solutions to elementary problems when the student is learning elementary methods, and it needs to *not* present answers achieved by methods that the student cannot understand.

2.2 Glass box

This means that you can see how the computer solves the problem. The program presents, or allows the student to construct, step-by-step solutions, not just answers. This is a very important point, and often misunderstood. It is comparatively easy for a program to get the one-line *answer* to a problem. It is more difficult to generate a multi-line *solution* with understandable steps. When combined with the requirement that the steps be cognitively faithful, this turned out to be not only the most important but also the most difficult criterion to satisfy. The glass-box criterion requires it to be apparent how each step was obtained; in practice this means accompanying each step with a "justification".

Figure 1 exhibits a sample step-by-step solution, captured from a *Mathpert* screen.

The simplest computer algorithms almost never generate cognitively faithful solutions. In applications where only the answer is of interest, and not the steps by which it was obtained, this doesn't matter. But in building software for mathematics education, it is crucial. "Human" solutions vary greatly with

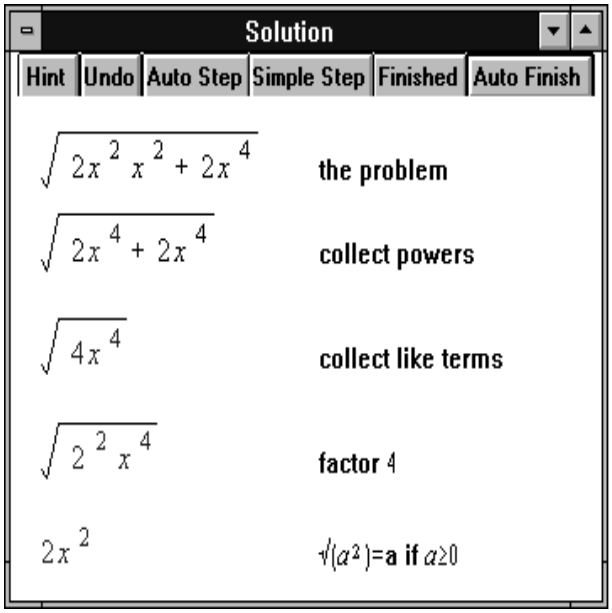


Figure 1: A sample step-by-step solution.

the features of each particular problem, delicately adjusting the choice and order of operations to be applied so that the resulting solution is economical and beautiful. Software for mathematics education will also have to generate economical and beautiful solutions.

In practice the ability of modern algorithms to solve problems the student cannot solve is not as large an obstacle to the use of existing computer-algebra systems as is the inability to break the computation into comprehensible steps. Even when a solution can be obtained by elementary means, it is important to break it into steps. For example, the following factorization of $x^{100} - 1$ can be obtained by a fairly short series of elementary steps, yet can be overwhelming if presented all at once:

$$(x-1)(x^4+x^3+x^2+x+1)(x^{20}+x^{15}+x^{10}+x^5+1)$$

$$(x+1)(1-x+x^2-x^3+x^4)(1-x^5+x^{10}-x^{15}+x^{20})$$

$$(1+x^2)(x^8-x^6+x^4-x^2+1)(x^{40}-x^{30}+x^{20}-x^{10}+1)$$

But if the computation system does not internally factor the polynomial by a series of elementary steps, but instead by a high-powered method such as Berlekamp's algorithm, there is no hope of getting it to generate the required series of elementary steps.

2.3 Customized to the level of the user

Cognitive fidelity demands that the software generate solutions that will be exemplary for the student. But students at different levels need different solutions.

A beginning algebra student needs a five-line solution to the common denominator problem $1/x + 1/y$. A calculus student evaluating a complicated integral does not want to see five lines devoted to $1/x + 1/y = (x + y)/xy$. It follows that the program must contain some, albeit rudimentary, representations of its user's knowledge. This representation must be sufficient to enable the generation of solutions with many small steps or few, powerful steps, depending on the requirements of the individual student.

It is apparently simpler to allow the user to create long ("fine-grained") or short ("coarse-grained") solutions, than to require the computer to do it. At least a "user model" would not be required. But there are still some difficulties even with user-generated solutions. Namely, this requirement means that the program must have a varied arsenal of mathematical operations. In order to allow the creation of fine-grained solutions, we need some very weak symbolic operations. *Mathpert* can generate a five-or-six line solution to the common-denominator problem $1/x + 1/y$. It can also generate a one-line solution. Figure 2 shows the five-line solution as it can appear on *Mathpert*'s screen.

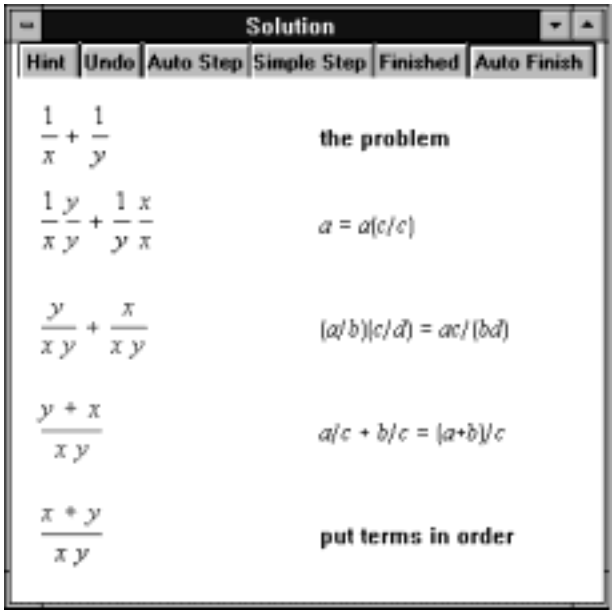


Figure 2: A solution with many small steps

Mathematica, *Macsyma*, and *Maple* can only generate the one-line solution, which is useless for learning common denominators. These general-purpose programs need only one simple operation for taking common denominators. To meet the requirement of generating fine-grained solutions, *Mathpert* needs several different operations, for use at different mathematical levels. While creating your solution, you can take big steps or little ones, as your ability and temperament dictate.

Buchberger [8] proposes a "white box/black box" model for symbolic computation software in education. A "black box" corresponds to a one-step solution, a "white box" to a fully detailed solution. Buchberger's "white boxes" are the same as my "glass boxes". But his discussion considers the relationship between the glass box requirement and the requirement that solutions be customized to the user. An intermediate case, where the solution is in several steps but not at the finest level of detail, could be analyzed in Buchberger's terms as a white box at the top level, in which subordinate steps are treated as black boxes. For example, when integrating rational functions, partial-fraction decomposition can be treated as a black box. *Mathpert* supports this paradigm fully, allowing the user complete flexibility to customize the level of detail desired in the solutions. It is a delight, when working out a step-by-step solution to an integration problem, to just push a button to get a partial-fraction decomposition. Figure 3 shows the beginning of a solution to an integration problem, in which two complicated algebraic steps are taken at a single mouse click.

2.4 Correctness

You cannot perform a mathematically incorrect operation or derive a false result with *Mathpert*. It is not as widely known as it should be that most systems in use nowadays do not have this property. I shall therefore give some examples.

In *Macsyma*, we can set `a = 0;` and then divide both sides by `a` using the command `%/a`. This will result in `1=0`, because according to *Macsyma*, $a/a = 1$ and $0/a = 0$. This example can be carried out in *Mathematica*, too, but you must first tell it that dividing an equation means to divide both sides. The cause of the error is ignoring the side condition $a \neq 0$ on the rule $a/a = 1$.

Here's another example: if $F'(x) = f(x)$ then $\int_a^b f(x)dx = F(b) - F(a)$. Apply this rule with $f(x) = 1/x$ and $F(x) = -1/x^2$ to get $\int_{-1}^{1}(1/x)dx = 0$. The cause of the error is ignoring the side condition that f must be continuous on $[a, b]$. This is a standard homework problem in Calculus 1.

When these examples are presented, the suggestion is sometimes made that these are simply "little bugs" that can be easily removed. This is a misconception. The root cause of these problems is that mathematical operations generally have "side conditions" which are logical propositions. For example, the side condition on the operation $a/a = 1$ is $a \neq 0$. When we make a mathematical calculation, each line is actually dependent on a certain list of assumptions.

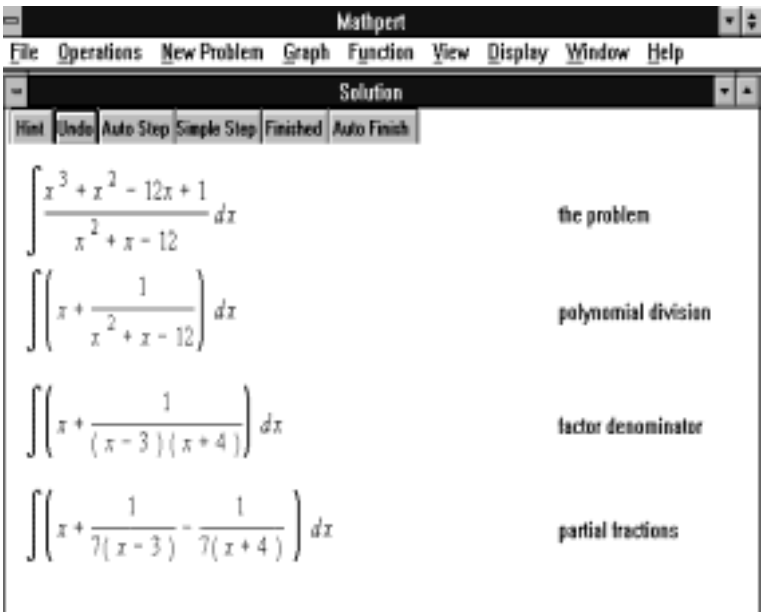
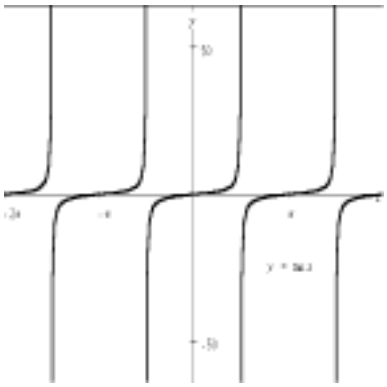


Figure 3: A solution with some steps using powerful operations

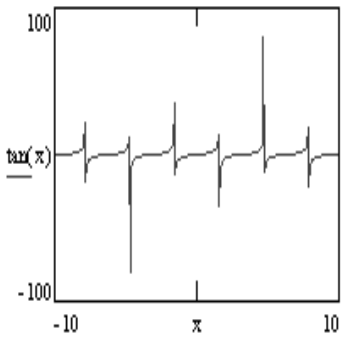
Some steps, for example replacing $\sqrt{xy}$ by $\sqrt{x}\sqrt{y}$, add new assumptions to this list. (In this case, the new assumptions are $0 < x$ and $0 < y$.) When we do mathematics by hand, we usually do not write the assumptions down explicitly, but sometimes we make a note of additions to the list, and maintaining the list correctly is vital, as the above examples show. Moreover, it is not possible to restrict the complexity of these assumptions. Denominators must not be zero; but denominators can be complicated, so the zero-sets of functions get involved. These often depend on the domains of other functions, so we need to be able to calculate domains. Domains, for example of $\sqrt{u(x)}$, will depend on the sign of u in a neighborhood. This kind of question can in turn depend on whether functions are monotone increasing or decreasing; things get arbitrarily complicated. The side condition for evaluating definite integrals already involves continuity of the integrand on an interval. These complexities show that the correctness principle cannot be "added on" to a system that was not designed from the start to support the maintenance of a list of assumptions during a calculation.

The correctness principle applies to graphs as well as to symbolic solutions. When graphs have singularities, most commercial graphing software gets most of them wrong. Graphs are made by connecting a finite set of points. These points will probably not hit the singularities exactly, so the graph may not even go off-screen at a singularity. Moreover, there is often a vertical line where the singularity belongs. This arises because the software just connects one point

to the next, and "knows" nothing about singularities. *Mathpert* avoids these pitfalls, by means that will be explained in Section 5. Figure 4 shows a graph of $\tan x$ made by *Mathpert*, and a graph made by purely numerical computation, as all other software I have seen does it.[2]



Mathpert graph of $\tan x$



$\tan x$ graphed by other software

Figure 4: Correct *vs.* incorrect graphs with singularities

Now that I've explained the difficulty of satisfying the correctness principle, let me also emphasize its importance. Although this paper deals with *educational* software, I find it disconcerting to think of flying on planes whose jet engines or flight-control systems were designed using mathematical software that can derive $1 = 0$ in two steps. If there are two-line contradictions, there may also be more subtle errors that do not stand out. Not every wrong answer will be obvious, even to an experienced engineer. However, let us restrict the discussion to education. Errors that are obvious to an engineer may not be obvious to a student. If the software draws graphs with vertical lines at singularities, for example, students may be easily confused.

2.5 User is in control

The user decides what steps should be taken, and the computer takes them. As in word-processing, the computer takes over the more "clerical" part. The principle that the user should direct the solution requires either a command-driven or mouse-driven interface. The principle of ease of use dictates a mouse-driven interface. When *Mathpert* operates this way, it is said to be in *menu mode*. The user selects an expression on the screen. This selection brings up a menu of operations. The user then selects an operation. These operations transform the current line and generate the next line. In this way a multi-line

[2]Some graphing calculators trap $\tan x$ as a special case and get it right, but they don't get $\tan x^2$ right, or even simple rational functions.

solution is built up. Earlier versions did not have the capability to select an expression from the screen, and a more extensive system of menus was used to select the operation; hence the name *menu mode*.

The principle that the computer can help you out when you are stuck requires there to be an automatic mode, or *auto mode*, in which the program can itself generate a solution. The user can switch between these two modes at will.

Each time I demonstrate *Mathpert*, and professors see the program generating step-by-step solutions to homework problems, some of them envision students simply having *Mathpert* do the homework for them while they are out watching *Beavis and Butthead*, and I have many times heard the question whether there is a way to deny the students access to auto mode. Indeed, the fear that students will learn little by watching the computer generate answers may be relevant if they are using software that generates *only* answers, and generates those answers by incomprehensible methods. The computer then becomes an oracle, and one who questions an oracle has no control.

In practice, students seldom misuse auto mode. They seem to take pleasure in finding and applying the correct operations. The fact that the computer carries out the mechanics of the application, so that you can't lose a minus sign or forget to copy a term, or incorrectly apply an operation, increases the user's control over the developing solution.

2.6 The computer can take over if the user is lost

A good symbolic computation program for education should be able to solve (upon request) all the (symbolic or graphical) problems commonly given in algebra, trig, and first-semester calculus. This is a very strong requirement, in view of the fact that what is required is not just the answer, but an economical, beautiful, and cognitively faithful step-by-step solution, tailored to the level of the student.

Even if we only wanted answers, however, it is important to note that the restriction to problems commonly given in courses is vital. There are technical results showing that it is impossible to write programs to solve *any given* problem of the types studied in these courses. For example, Richardson [18] showed that it is impossible to write a program that will take as input a given identity $f(x) = g(x)$ in one real variable, and determine whether or not that identity is valid, where f and g are built up from exponentials, logs, and trig functions using the arithmetic operations.[3] Nevertheless, we do implicitly teach methods, not just a collection of specific examples, and the requirement is simply that the program must embody all the methods we teach.

[3] That is, there is no *decision algorithm* for this class of identities. Normally in trigonometry, the trig functions are not nested, and the arguments of the trig functions are linear. There is room for more careful work, narrowing the gap between the large classes of identities for which there is provably no decision algorithm, and the small classes for which we actually possess a decision algorithm. The "boundary" is at present largely unexplored.

The ability of the system to generate an "ideal" solution can be put to good pedagogical use to assist a student who is having difficulty. Several such features are included in the design of *Mathpert*, but as these are not fundamental design principles, they will not be discussed in this section.

Problems often can be solved in more than one way, for which the jargon is "multiple solution paths". Of course the ability of the computer to generate a single "ideal" solution does not contradict its ability to support multiple solution paths in menu mode. But it actually offers good support for multiple solution paths in auto mode too. Although the system generates for each problem a unique "ideal" solution, it has to be able to do that from any point. The user could take one or several steps, and the computer must be able to finish up the problem from there. Usually the computer can be sent down a particular "solution path" in a few manually-directed steps, after which it will finish the solution along the desired lines.

Sometimes the necessity of solving a problem by trial and error is brought up in this connection. *Mathpert* can carry out such a search, as when searching for the factors of a quadratic polynomial. When this subject is first being learned, the attempted factorizations are shown to the student. There is a more extensive discussion of trial-and-error in Section 3.4.

2.7 Ease of use

The system must be EASY TO USE. This point has been emphasized over and over by everyone who has considered the use of computers in mathematics education. Students are likely to be poor typists and may be afraid of computers. Many of them are also afraid of mathematics. It is essential that the students feel the computer is a help, not just another hurdle. Reports I have seen of many experiments with computer-assisted calculus instruction show that often the students do feel the computer adds to their difficulties, rather than helps. The day has not yet arrived when many students buy mathematical software that is not required, because they think it will help them learn.

Ease of use is often considered to be an "interface issue", but I don't believe that a well-designed program for education can really be separated into an "interface" and a "kernel" operating completely independently. Take the generation of step-by-step solutions, for example–is that an interface question? If you take the narrow interpretation, it is not. Only the means by which the user commands the steps to be taken, and the means by which the results are displayed or printed, are interface questions. But on this narrow construction of the term "interface", the question of ease of use cuts much deeper than the *mere interface*. Ease of use means that it is easy for the user to accomplish a given purpose. Whether the purpose can be accomplished easily, or at all, may well depend on the capabilities of the program as well as on the design of the menus and dialog boxes and which buttons can be clicked when.

A critical moment for "ease of use" comes when the user is "stuck", that

is, does not know how to proceed. If the system has the capability to solve the problem, it can be viewed as an "interface issue": *how to use that internal capability to best advantage to help a student who is stuck*. Whether one has the program simply exhibit the next step, show the next menu choice, or give a hint in English (or French for that matter), is a question of interface design, in my view. I would not limit the term "interface" to mean where you have to click or what you have to type to cause these things to happen.

Nevertheless, questions of interface design even in the narrow sense are of some importance. I will consider some of them in Section 7.

2.8 Usable with standard curriculum

Even if the professor isn't using it, and knows nothing about it, it should be usable by the student. This criterion is perhaps more controversial than my other criteria. But I have always believed that serious curriculum change will be driven by student use of software, rather than the other way around. After a decade of well-funded initiatives to bring technology into the calculus classroom, most classes at most universities, and all classes at many leading universities, are still taught without software assistance, and with a substantially unchanged curriculum.

Answer-only programs such as *Mathematica* and *Maple* are hard to use with the standard curriculum in mathematics, which emphasizes step-by-step solutions. This has led some people to call for a new curriculum, in which students would not learn the traditional step-by-step solution methods. Methods of integration in particular have suffered a bad press. There is talk of emphasizing *concepts* instead of *techniques*. Of course, nobody is against teaching mathematical concepts–it's like motherhood and apple pie. But there is a serious question whether it makes sense to teach concepts without techniques.

Given the fact that answer-only software can't support the present curriculum, the faculty has been left with the choice: stick with the old curriculum (and no software), or take a big leap: changing both what is taught and how it is taught at the same time. Change will be easier if they can change gradually, adding software support to the existing curriculum, and then adjusting the curriculum later as required.

3 Cognitive fidelity and glass box in *Mathpert*: operations and pedagogy

The achievement of cognitive fidelity in *Mathpert* depends on the implementation of "operations" that correspond to identifiable "steps of computation" taken by a student in the course of solving a problem. These operations can then be invoked by name or formula from a menu (in various ways) or applied by the program itself when generating an exemplary solution. It is essential

that the basic steps of the solution correspond to what the human perceives as "operations".

3.1 Design of the operations

The cognitive fidelity principle requires a careful and detailed analysis of the subject matter, to determine the correct choice of operations. There are some 130 menus of up to sixteen operations each, so there are about a thousand different operations that *Mathpert* can perform (not counting graphical operations). I believe that these operations permit the solution of all (non-word) problems normally taught in algebra, trig, and calculus. In many cases it was a non-trivial task to construct the appropriate set of operations, though in some cases it was not difficult for an experienced teacher.

3.2 Rewrite rules, matching, and pedagogy

A rewrite rule is a one-directional equation, such as $a(b+c) = ab+ac$. By saying it is one-directional, I mean that you can use it to rewrite $a(b+c)$ as $ab+ac$, but not the reverse. A rewrite rule is applied by using a process of *matching* to see that some complicated expression "has the form of" the left hand side. For example, we can use the rule just mentioned to rewrite $x^4(x+1)$ as x^5+x^4. Purists will note that I have used some other laws of algebra and arithmetic as well!

Although rewrite-rule technology at first sight appears attractive, many of the operations cannot be expressed in rewrite-rule form, because they take an arbitrary number of arguments and because other arguments can come in between. Consider collect powers, for example, which is related to the rewrite rule $x^n x^m = x^{n+m}$, but applies to $ax^2bx^3x^4cx$. The last x has to be collected even though it doesn't have an explicit exponent. The other x's have to be collected even though they are separated and there are more than two of them. Moreover, there are further difficulties with rewrite rules in the presence of associativity and commutativity.[4]

[4]Readers with training in the relevant branches of logic or computer science will be aware that there is a vast literature on rewriting techniques. This technical footnote is directed to such people (there was one among the referees). One of the general aims of this research is to provide improved matching (technically: unification) algorithms which will incorporate algebraic laws into the matching step, so that for example $ab+c$ should match $c+ba$. Although there are beautiful algorithms and theorems about those algorithms, it should be noted that the best of the crop will still not handle the rewriting of $x^4(x+1)$ as x^5+x^4. Readers wishing to explore these algorithms will find an initial discussion and further references in [9].

This body of theory has not found application in *Mathpert*, and not because the author was ignorant of the theory. Nevertheless *Mathpert* successfully handles matching. For example, it is able to apply the rule $\cos^2 x + sin^2 x = 1$ in the context $2+3\sin^2 5x - 5xy + 3cos^2 5x$. But if the second $5x$ is made $x5$, *Mathpert* will miss the match until the operation Order factors is applied. Note, however, that associative-commutative unification will miss the match in the first example until the 3 is factored out.

These observations have implications of interest to both programmers and educators. The implication for the programmer is simply that rewrite-rule technology isn't enough: each operation has to be implemented as a function in the programming language.

The implication for educators is more far-reaching, though less obvious. Many of the difficulties ("bugs") that I have observed in student solutions (or failures to generate solutions) can be attributed to failures in the student's matching algorithm.[5] Just to give one example: consider using the (reverse) distributive law $ab+ac=a(b+c)$ to rewrite $x^4(x+1)+x+1$ as $(x^4+1)(x+1)$. Even if the difficulty about grouping the first $x+1$ so as to match a to $x+1$ is overcome, there is still the problem that c is 1 here, so literal matching won't suffice. The matching procedure itself has to incorporate some other algebraic laws for this to work right. Another, unrelated difficulty apparent in this same example occurs the first time students have to match one of the letters in a pattern to a sum, instead of to just a monomial. Nearly every student fails the first time this is asked. Yet many textbooks I have examined fail to give an illustrative example. The first such problem occurs in the homework assignment, where it is guaranteed to be frustrating. Many a case of "math phobia" begins with examples like this, in which a homework problem is asked which in some sense "follows from" the principles in the text, but involves some new twist, which the computer scientist can describe precisely in terms of extra code required for the matching algorithm.[6]

3.3 Order of operations

It is interesting, both for the computer scientist and the educator, to consider the problem of which rule to apply when. First of all, one should observe that many rules are inverses: for example $\log ab = \log a + \log b$ can be used in either direction, and indeed *must* sometimes be used in one direction, and sometimes in the other direction. Sometimes we must factor, sometimes we must multiply out.

The simplest control mechanism would be context-free, in the sense that the decision as to which operation to perform could be made *locally*. For example, the decision whether to put a sum of fractions over a common denominator would have to be made without regard for the context in which the sum of fractions occurs. In practice, there are two separate factors influencing such decisions that are not directly available in this method. First, it matters what kind of problem we are solving. For example, if the problem type (the goal of the problem) is common denominators, obviously we want to use common denominators regardless. If the problem type is partial fractions, equally obviously

[5] The "buggy theory of learning" was first promulgated by J. S. Brown and R. Burton, see for example [10].

[6] I am certainly not the first to observe that student errors are sometimes attributable to faulty matching. See for example [14] and [16].

it would not be a good idea. Second, the context of the term being simplified (in the sense of the current line in which it is embedded) matters: for example, we never want to use common denominators inside an unevaluated derivative. It looks silly to use common denominators inside a derivative, like this:

$$\frac{d}{dx}\left(\frac{1}{x}+\frac{1}{x^2}\right)$$

$$\frac{d}{dx}\left(\frac{x+1}{x^2}\right)$$

followed by a long computation using the quotient rule; the "correct" solution differentiates term-by-term, without using common denominators. There are many instances of such contextual dependencies.

Adjusting *Mathpert's* problem-solving algorithm to take these contextual demands into account has accounted for much of the work on the algorithm. None of this would be necessary in an answer-only program. On the other hand, a great deal of fine-tuning is required to cause the algorithm to use the right operations in the right order to produce the right step-by-step solutions. During the process of developing *Mathpert*, I observed many loops and infinite regresses generated by meta-rules that conflict in a given situation. Each time I had to modify the meta-rules which looked good before, adding or changing the conditions under which certain mathematical operations should be applied.

Loops. Another complication, in addition to contextual dependencies, is the possibility that different operations taken in combination can cause loops. For example, you can factor and then expand a polynomial; so auto mode must never activate both operations, or a loop will result. There is nothing like a formal proof that the current system is always terminating; such a proof would be extremely complicated and probably could only be carried out with mechanical assistance. Indeed, it may well not even be the case–there may well be loops that haven't been observed. Certainly I have observed many unanticipated loops in the process of development and testing.

In order to guard against unanticipated loops in automode, *Mathpert* incorporates automatic loop detection: it will never go into an infinite regress, but will detect that it is about to enter a loop. Early versions would then deliver a rueful message: *I seem to be stuck in a loop. You'd better take over.* The present version does something more like what a person would do: it just doesn't use the operation that would cause a repeated line, but looks for something else to do.

Avoiding Loops in Equation Solving. The avoidance of loops is particularly critical in equation solving. The difficulty is that various algebraic laws have to be applied in one direction for some equations, and in the reverse direction for other equations, so that at first sight it seems impossible to avoid loops and still solve all equations. Here I have had some help from Alan Bundy's theory of equation-solving [9]. His general theory had to be integrated with special-purpose solvers for linear equations and non-linear polynomials, but his ideas

of "attraction" and "collection" guided the way auto mode selects operations according to the form of the equation.

3.4 Pedagogical implications

Once algebra, trig, and calculus have been codified into a set of about a thousand rules, can we say, *if you know the rules, you know the mathematics*? This is true to the extent that all the symbolic problems in the books can be solved using those rules. But it overlooks the fact that you must not only know the rules, you must know which rule to apply when. This is really much the more difficult thing to know. *Mathpert* has several thousand lines of what amount to *rules for using the rules*. These rules are internal to the program, rather than visible to the user, and govern the machinery of the automode solution-generator. The process of developing and fine-tuning these meta-rules took much more time than the implementation of individual operations for computation, which was comparatively simple.

Yet, when we look at textbooks and course outlines, most of what students are taught is how to execute the rules one or two at a time. Each section presents another few rules, and exercises which are solved using those rules. The student studies algebra this way, and then learns trig a few rules at a time. When the student comes to trig identities, for the first time a variety of different rules is required. A detailed analysis by A. Lauringson revealed 59 operations required to solve a certain set of identities, while corresponding sets on other topics required at most 30 operations. Probably the notorious difficulty of this subject is mainly attributable to this feature. There is little or no discussion in textbooks about strategy for trig identities. For example, there are three rules for expanding $\cos 2x$, and success is often dependent on choosing the right one, but textbooks do not even discuss the issue.

Solving equations is another area in which strategy is crucial, but seldom explicitly discussed. Here is a place where the contributions of computer science should be allowed to "trickle down" to textbooks. Bundy's ideas of attraction and collection are simple and straightforward and can easily be explained to algebra students. I am resisting the temptation to prove this point by explaining them here and now; but see [9]. These ideas bring order to the otherwise mysterious business of solving equations, reducing what appears to be an art to a science. Don't write another algebra textbook without reading Bundy.

Then the student goes into calculus, where he or she is required to "simplify" expressions in order to solve calculus problems. One semester I kept statistics on exam errors: Eighty percent of exam errors in first-semester calculus were actually errors in algebra or trig that theoretically were prerequisite to the course. These errors were sometimes due to failures in the matching algorithm, sometimes to incorrect memory of a rule pattern, but often simply to not recognizing which rules would do the required job. *That* was never taught; the "born mathematicians" pick it up somehow, by osmosis.

The above remarks should be carefully considered not only by designers of such software, but especially by textbook authors, course designers, and teachers. No wonder students are confused: the control apparatus is more complex than the rules being controlled, but is never explicitly discussed! Textbooks should contain explicit discussions of these matters, as well as carefully chosen examples and exercises to illustrate the correct choice of operations according to the goal and the context. Most important, they should contain many more sections and examples in which a mixed bag of different rules need to be applied.

One interesting question is this: how much does mathematics depend on trial-and-error search? My experience in developing the auto mode algorithm of *Mathpert* leads to the conclusion that search is used in elementary mathematics in only a few places: searching for the factors of a quadratic polynomial, searching for a linear factor of a higher-degree polynomial, and searching for a good grouping of the factors in factor-by-grouping. In calculus we might add, searching for a substitution in integration by substitution, and searching for a choice of parts in integration by parts. All other problems, including the choice of which rule to use on $\cos 2x$, can be solved without searching.[7]

One of the referees raised the possibility that *Mathpert* could teach or give access to its internal meta-rules. This suggestion has two answers. The short answer is that *Mathpert* is not designed to be explicitly tutorial (see Section 1). It would almost certainly be a good idea to give more instruction in problem-solving strategy, possibly aided by more explicit "rules of thumb" suggested by the internal *Mathpert* meta-rules. However, it is not within the scope of the statement of purpose of *Mathpert* given in Section 1. The long answer is more interesting, and bears on deep issues in artificial intelligence and cognitive science. Namely: it is not at all clear that human mathematicians really operate in this way, using a collection of meta-rules. Maybe there is planning and goal-seeking at a deeper level; maybe there is "understanding". Chess programs operate by rules and calculations, but perhaps grand masters do not. Even if this is true, students of chess do study rules. This issue has caused more arguments than any other in the philosophical foundations of artificial intelligence.

4 Customizing the solution in *Mathpert*

In Section 2 we have given an example to illustrate the fact that *Mathpert* can generate solutions with many small steps, or solutions with few but powerful steps. This section will discuss how this is achieved.

First of all, when the user is controlling the steps of the solution, the user can choose the operation, and so it will suffice if there is a choice of the powerful or the weak operation. When the user has selected a sum containing a fraction

[7] Nicaud *et. al.* [16] describe a system that helps students learn to solve factorization problems by teaching them to search for a solution, backtracking at failure. The examples given in the paper are in the topic known in Algebra I as "factor by grouping".

or fractions, the menu of available operations will provide more than one choice. For the beginning algebra student (that is, if the problem was entered under an elementary topic), this list will include Find common denominator. This will simply multiply both fractions by an appropriate factor c/c, as in the first step in Figure 2. If the user thoroughly understands common denominators and basic algebra, for example a second-semester calculus student, the operation Common denominator and simplify numerator will do everything: for example,

$$\frac{1}{x^3+1} + \frac{1}{x^2-1}$$

will be transformed to

$$\frac{x(x^2+1)}{(x-1)(x+1)(x^2-x+1)}.$$

A student still wanting to see more detail can first explicitly factor the denominator, and then use Common Denominator, and finally Simplify, to get a three-step solution.

Customized solutions like this are easy for *Mathpert* to provide in menu mode, because the symbolic computation engine was developed in accordance with the design principle of cognitive fidelity. Another project whose progress I have followed, was originally based on the idea of putting a step-by-step interface on REDUCE. After the implementation had progressed for some time, it was noticed that the steps were often too large and confusing. Note that this is a completely different class of difficulties from the logical problems discussed in connection with the correctness principle. Once one starts down the road of implementing simpler operations in terms of more complex ones, using a programmable CA system such as REDUCE or *Mathematica*, what guarantee is there that the process will be simpler than implementing an educational CA system from scratch? And even if it turned out to be, one would still have the logical difficulties.

Mathpert is designed not only to permit the user to produce different solutions to the same problem, but also to do so itself when running in auto mode. If a common denominator is required to solve an integral, it should be done in one step for a calculus student, but for a beginning algebra student, automode should produce the long solution to $1/x + 1/y$. How is this to be accomplished?

In order to support the generation of different solutions to the same problem, *Mathpert* maintains an internal model of its user. This model has been discussed in some detail [4]; for this paper it will suffice to know that the model consists essentially in labelling each of the approximately one thousand operations as either *unknown*, *learning*, *known*, or *well known*. Initializing the model consists in assigning these labels.

Experience since 1990 has taught an important lesson. The original intention, inspired by the "buggy model of learning" [10] was that the user model

should model a particular individual. This model was to be initialized by a diagnostic test program, and the model was even made user-editable in early versions of *Mathpert*. However, experience soon showed that it is easy to get an "out-of-balance" user model, which will cause strange-looking solutions, in which there may be several tiny steps and suddenly a mysterious large step. Indeed, reflection shows that we don't actually want to model the *real* student, but rather some fictional *ideal* student at the appropriate level for the subject, so that the auto-mode solutions are ideal solutions, rather than ones which might be generated by our (buggy) real student.[8]

The end result of these considerations is that the user model has become invisible to the user. When the user first enters *Mathpert*, he or she chooses from a menu what the "topic" of the session will be. These topic choices are intended to encompass about one day's lesson each (corresponding to one section of a textbook). The user model is automatically initialized based on the choice of topic. This user model will produce solutions at the level of detail generally appropriate for a class on that topic; the solutions will not use operations that are unknown (unless the problem can't be done without them). Thus L'Hôpital's rule will not be used (in auto mode) on $\lim_{x\to 0}(\sin x)/x^2$ when L'Hôpital's rule won't be taught until next week; but next week, when the topic is L'Hôpital's rule, it will be used.

In short: it suffices to customize the solution to *the level of* the individual student. It doesn't really need to be customized to the *individual* student.

5 The correctness principle in *Mathpert*

Necessity of a theorem-prover. Having established the difficulty and importance of the correctness principle, let me remark on what it took to build a system that satisfies it. *Mathpert* incorporates a non-trivial theorem-prover (about 6000 lines of C in the current implementation, about ten percent of the total code). This prover is a piece of work that can stand on its own as a contribution to the branch of computer science known as automated deduction.[9] There are few other programs combining some symbolic computation capabilities with logical deduction capabilities; we may mention Wu's work on geometry [19] and *Analytica* [11] in this connection. Descriptions of the *Mathpert* prover can be found in [3] and [5]; and a discussion of some later additions is in [6].

Logic is in the background. Although a correct handling of logical matters is essential for correctness, we rarely ask students to consider logical matters

[8] A team working on the APLUSIX project has developed a program [14], [15] which diagnoses a student's knowledge in a limited sub-domain of algebra, based on some twenty or thirty transformation rules in polynomial arithmetic. This work represents the state of the art in diagnosing and modelling individual students in the subject of mathematics.

[9] I have recently written another program called *Weierstrass*, based on the *Mathpert* prover, which has automatically generated an epsilon-delta proof of the continuity of x^3; previous programs could do this only for linear functions.

explicitly. The cognitive-fidelity principle then requires that the logical work performed by the software remain mostly invisible. The glass-box principle, however, requires that it be visible on demand. Each (visible) line of the solution has associated with it a list of assumptions on which it depends. For example, if you enter $1/x + 1/y$ as the problem, this will depend on the (implicit) assumptions $x \neq 0$ and $y \neq 0$. *Mathpert* makes these assumptions explicit, but does not display them unless you request it.

One of referees questioned this point of the design, saying that one of the main points of undergraduate mathematical education is training in logical thinking, and that therefore attention should be focussed on the logical aspects of the steps. Personally, I think this is a valid point deserving of consideration by teachers and textbook authors. Indeed the View Assumptions option in *Mathpert* can certainly be used to expose the logical underpinnings, so the teacher could simply instruct the student to make use of it. Moreover, on those occasions when *Mathpert* refuses to execute an operation because its side conditions cannot be satisfied, the user's attention will be forcibly drawn to the logical aspect of the situation. But the design principle of supporting the existing curriculum took precedence in this case (as well as some others) over the desire to "improve" the existing curriculum according to my own lights.

Correctness of Graphs. As discussed in Section 2, a purely numerical approach to graphing leads to incorrect graphs when the function being graphed has jumps or singularities. Graphs are made by connecting a finite set of points. If these points are chosen simply by partitioning the x-axis, the points will probably not hit the singularities exactly, so the graph may not even go off-screen at a singularity. Moreover, there is often a vertical line where the singularity belongs. This arises because the software just connects one point to the next, and "knows" nothing about singularities. Similarly, jumps will be drawn as short nearly-vertical lines. There is another class of function that is often incorrectly graphed: those with many maxima and minima close together. Two common examples would be $\sin(40x)$ and $\sin(1/x)$. What happens to a purely numerical grapher is that when the peaks are sharp, a maximum will occur between two plotted points, and the line segment connecting the points will simply cut across, going nowhere near the true maximum. Even an "adaptive step size" algorithm such as is used by *Mathematica* will not help, because the numerical clues that the step-sizer looks for occur only near the peak, and they are skipped too! Matters are made worse by the fact that these errors sometimes produce highly symmetric patterns, e.g. in the case of $\sin ax$ for a between 30 and 50. It's an interesting puzzle to explain the patterns, but they bear no relation to the true graph, and may well confuse a student.

Mathpert solves these difficulties by making use of its symbolic computation engine both while preparing to graph and during the actual graphing. Let us consider the problem of sharp maxima first. First of all, before graphing *Mathpert* computes the symbolic derivative of the function to be graphed, and then as it graphs, it numerically evaluates the derivative as well as the function.

If the derivative changes sign from one plotted point to the next, *Mathpert* uses a numerical equation-solving method to find a zero of the derivative between the two points, and makes sure to plot a point at that x-coordinate as well as the original two. In this way it ensures hitting the maximum. Of course, if the graph has maxima closer together than one pixel, some inaccuracies are going to result anyway. After all, there is no hope of drawing a perfect graph of $\sin(1/x)$ on a screen with a finite number of pixels! But what we can achieve is a clear picture of the envelope of the graph.

Turning to the problems of singularities and jumps: *Mathpert* appeals to the symbolic code used by the theorem-prover (not the operations accessible to the user) to calculate the singularities (if any) before making the graph. It then takes care to draw the graph correctly at the singularities, now that it "knows" where they are. For example, it calculates that the singularities of $\tan x$ are at $(2n+1)\pi/2$, and then when graphing $\tan x$ over a specific interval $a \leq x \leq b$, it calculates the relevant values of n and draws the graph correctly.

In difficult cases the program may fail to find a formula for the singularities. Even if it does find a formula for the singularities, if that formula involves an integer parameter as in the above example, it may be too difficult to find the set of relevant values of n. That set may not even be finite. For example, $\tan(1/x)$ is too hard. But, *Mathpert* at least graphs all rational functions correctly, and I hope it graphs all functions that would arise in a calculus course correctly.

It is easy to adapt the result of Richardson [18] to show that the problem of computing the singularities of a given elementary function is not recursively solvable. No matter how one improves the algorithm, there will always be functions whose singularities are not calculated. However, in such a case the user will get a warning that *Mathpert* could not calculate the singularities, and the graph may be incorrect.

Another approach to these problems, "honest plotting", is reported on in [1], [2]. "Honest plotting" refers to the use of interval arithmetic in calculating the graph values. That is, all arithmetic is done carrying along upper and lower bounds. In places where the possible error is significant, the graph line gets thicker. In bad examples, such as $\sin(x)/x$ near the origin, it becomes a region instead of a line, but at least the correct graph is known to lie in the shaded region. Honest plotting was described years ago in [12] and is used in Avitzur's Graphing Calculator (which is distributed with the Power Macintosh computer).

The Correctness Principle and Pedagogy. I found in the writings of Maria Montessori, the Italian educator, a principle that is highly relevant.[10] Montessori demanded that educational materials satisfy the principle she called "control of error". This means that the materials should be "self-correcting": the materials themselves must inform the student, without the intercession of the teacher, when the activity has been completed successfully. Montessori had in mind physical materials for preschool and elementary school children. For ex-

[10] See Chapter 24 of Montessori [1967].

ample, suppose the task is to sort ten wooden dowels by length. Control of error can be provided by making the rods of different diameters, and requiring the sorting to be done by inserting the rods in holes. You can't put a rod in a too-small hole. You can, of course, put a rod in a too-large hole, but if you do, you'll have some rods left over that won't fit anywhere. There is only one way to fit all the rods in a hole, and that's the correct way.

Mathpert provides control of error by means of the correctness principle. You simply can't take a mathematically incorrect step. You can, of course, take a mathematically correct but irrelevant step; you can factor a polynomial and then multiply out the factors again. You can go on repeating those steps all day long if you like, but you can't make a mistake when multiplying out the factors. When the problem is finished, *Mathpert* will tell you *That's the answer.*

6 Using the computer's power when the user is stuck

One of the design principles in Section 2 required that the computer should be able to take over on request (presumably, that means when the user is lost). There are, however, a number of intermediate and pedagogically interesting states the user can be in besides "proceeding competently" and "lost". There is, for example, the state "now what?", in which the user has been proceeding competently, and is generally competent with this topic, but does not see immediately what to do next. Even the state "lost" can be divided into "lost and given up" and "lost, but hoping to recover".

How can we best use the power of the computer to solve the problems, when the user is in these various states of confusion? This section will explain how the current version of *Mathpert* uses its auto mode capabilities in different situations.

First of all, the user who is "lost and given up" can simply click the Auto Finish button to let the computer finish the solution. Presumably the lost student then studies the solution, in order to do better next time.

The user who is "lost, but hoping to recover" can click Auto Step. This causes *Mathpert* to generate just one (more) step of the solution. Repeated clicking on Auto Step will thus duplicate, one step at a time, the effect of Auto Finish. But perhaps the student can realize his hope of recovery after seeing one or two steps, and continue the solution himself.

Hint generation. A more interesting case is the user in the state "Now what?", who does not quite feel lost, but can't quite figure out what to do either. This user could of course click Auto Step, but may be reluctant to "admit defeat" by so doing. *Mathpert* provides another alternative: the Hint button. When the user presses Hint, *Mathpert* internally generates one more solution step, but does not show it. Instead, having determined what operation

it would use, it looks up an appropriate natural-language hint in a pre-stored table of hints. These hints are designed to sound like what a teacher would say in the situation. They are conversationally phrased, containing formulas only when necessary, but they are designed to enable the student to extract the menu choice which should be made to invoke the suggested operation, or at least the term to select to proceed using the Term Selection Interface.

Error Analysis. Although you can't carry out an incorrect step in *Mathpert*, you can try to do so, by making an inappropriate menu choice. Of course, the operation may be applicable even though in some sense inappropriate, in which case *Mathpert*, in accordance with its principles, will let you take that step. But you may have chosen an inapplicable operation, in which case *Mathpert* must refuse your request. It is desirable that in this situation the most informative error message possible should be supplied. Of course, some very common errors can be trapped individually, and appropriate messages can be stored in the program. However, with over a thousand operations, there are a million possible pairs of the form (*correct operation*, *chosen operation*). So some attempt at dynamic error-message generation is imperative. For example, some errors are simply due to omitting a necessary preliminary step. *Mathpert* will catch many of these errors. The method is this: when an operation is inapplicable, *Mathpert* internally takes four steps of the automode solution starting from the current line. If the user's operation is used in one of these four steps, it is a good bet that the user is "on the right track", and a helpful message can be generated informing the user what preparatory step has to be done first. Of course, errors that are "not in the ballpark" will still have to generate some uninformative message such as *Sorry, that operation can't be applied here.*

7 Traditional interface issues: ease of use

In this paper I have focussed attention on the way in which the interface and "kernel" or computational engine of a good symbolic computation program for education are *necessarily* interrelated. Nevertheless, there are a number of issues, mostly connected with my design principle *Ease of Use*, that are traditional "interface issues".

7.1 Source of problems

Although *Mathpert* can solve problems given by the user (instead of only pre-stored problems), I found when our student laboratory opened that nobody actually wants to type problems in. Both students and professors want to find the problems already in a problem file ready to call up. Moreover, if using the program in connection with a class, they want to find the exact problems that have been assigned in class, not the problems chosen by the author of *Mathpert*. This is important because it permits professors to make the use of

Mathpert optional. This problem has been solved, in a way that gets appropriate problems to the student without even requiring file names to be known or typed, while still allowing full flexibility to either student or professor to customize the problem files as desired.

When a student installs her new copy of *Mathpert*, she will choose her textbook from among the supported textbooks, and the problem files for that textbook will be included in the installation, in addition to the problem files developed and distributed with *Mathpert*.[11] Nevertheless, the program has to support entering an entirely new problem of your own choice. At present this means typing things like `x^2+y^2` when you mean x^2+y^2. There is thus a "one-dimensional notation" for typing in mathematics, different from the "two-dimensional notation" used in books and for on-screen display. Some other programs offer "equation editors": click-and-drag interfaces, using palettes of symbols, that allow users to build up formulas directly in two-dimensional notation. The construction and desirability of such interfaces is today a widely-discussed issue, to which I have little to contribute. What little I have is simply this observation: while an engineer is likely to bring her own problem, a student is likely to want to solve the homework problems. If they are already on disk, nobody will have to type (or click and drag) to get the problems entered. Figure 5 shows what it looks like to see the problems stored on disk. You can press Next or Previous to peruse the available problems.

7.2 Selection of operations

There are over a thousand operations in *Mathpert*. Even with a menu-driven interface, if all thousand operations were always available, it would be confusing to try to find the one you need. There is therefore a serious "interface problem" in enabling the user to find an appropriate operation and cause it to be used. It seems useful to distinguish here between the user who *knows what she wants to do*, and the user who does not. Let us consider the user who knows what she wants to do, by which I mean that she knows which operation she wants to apply. This is not the same as knowing the name by which this operation is known on a menu; what is meant is that she could easily write out the next step using pencil and paper, unless the computations are too intricate to do by hand.

In particular such a user will know which sub-expression she wants to work on. *Mathpert* permits her to *select* this expression by using the mouse to enclose it in a rectangle. When she makes this selection, the enclosing rectangle gets a colored background, just as when you select text in a word processor, and a short menu of operations comes up, sometimes containing only one operation, but sometimes containing three or even ten operations. The listed operations include

[11] As of March 1995, only two textbooks are supported, and we are in the process of obtaining permission from publishers to support more.

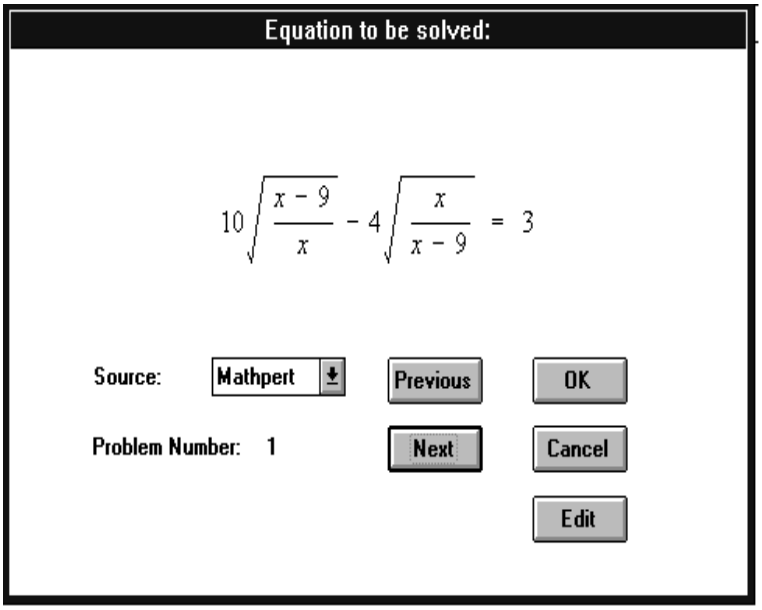


Figure 5: Getting a problem without typing

everything you could do *to* the selected term, and the include also everything you could do *with* the selected term, such as transfer it to the other side of the equation. This *Term Selection Interface* allows the user who knows what she wants to do to get the next line on the screen as quickly as possible, with a minimum of menu-searching. Figure 6 shows how the selected term appears on the screen. The selected term is highlighted (the default color is light yellow, but all colors can be changed by the user, and of course the illustration here is monochrome). The menu of possible operations should appear at the right of the selected equation, but the image-capturing software used to make the illustrations proved incapable of handling a "floating menu". Therefore the menu has just been typeset, rather than captured from the screen.

The term selection interface provides an elegant solution to the operator selection problem for the user who knows what she wants to do. Now consider the user who does not know what she wants to do. This user has several options, among them Auto Step and Hint, which have already been discussed. There is a good chance that the use of Hint may convert this user into one who can profitably use the Term Selection Interface. If the user doesn't feel lost enough to press Hint, there is an intermediate option: go browsing in the menus of operations. Before the implementation of the Term Selection Interface, all users had to do this to take the next step in menu mode. This browsing is not quite so difficult as it might seem, as measures are taken to cut the number

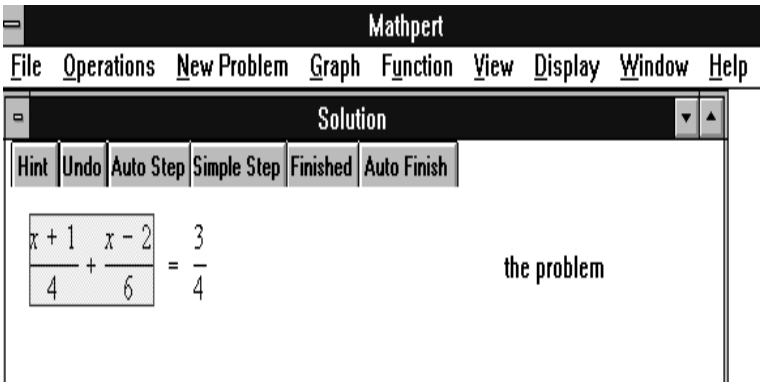


Figure 6: Selecting a term to work on

subtract from both sides
divide both sides
collect all $\pm$ terms in a sum
cancel $\pm$ terms
write it as a polynomial in ?
add fractions $a/b \pm b/c = (a \pm b)/c$
common denominator
common denom and simplify numerator

Figure 7: The menu brought up by the selected term in the previous Figure.

of menus of operations that can be browsed to a minimum. Luckily, in any given mathematical situation, most of the operations are irrelevant. Normally, all the operations you might need will fit on fewer than 20 menus of at most 16 operations each, so you need only one menu with entries like Fractions, Square Roots, and Integration by Substitution; each of those has an associated pop-up menu listing the operations themselves. Of course, this menu of 20 menus must be dynamically computed from the problem type and the current line of the computation–it can change after each new line. For example, when the last integral sign is gone, no more integration menus will appear.

The Term Selection Interface is quite new, and all laboratory experience with *Mathpert* was based on the menu-browsing interface. Two facts emerged: (1) It is feasible to use this interface, indeed students quickly develop the skill of finding the operation they want. Nevertheless, (2) Nobody really likes doing so, although they do like seeing the solution develop as a result. In other words, the menu-browsing is a psychological "cost" to be weighed against the "benefit" of results obtained. The Term Selection Interface, on the other hand, is fun to use.

Incidentally, multiple selections are allowed; for example one can move several different terms to the other side, or one can apply the same operation to two different subexpressions at the same step.

7.3 Arguments of operations

Some operations require an argument, that is, an additional term to be supplied by the user. For example, the operation add to both sides, used in solving equations. If the user chooses this operation in the old menu-browsing interface, a popup window will appear for data entry, and she would be prompted *Add what to both sides?* After she enters a term (and it is checked and accepted as a suitable response for that particular operation), then the popup window will disappear and she can continue. In auto mode, if an operation requires an argument it must be automatically selected.

The Term Selection Interface improves this markedly: to multiply an equation by something, just select that something in the visible equation, and then choose Multiply both sides from the resulting short menu. This works fine as long as the desired argument is visible somewhere to be selected. In some cases multiple selection can be used too, as in the case of multiplying an equation by 12 because it has denominators of 3 and 4 in different places: just select both the 3 and the 4. But if for some reason you want to multiply the equation by 27 when no 27 occurs, you will have to do that the old way, by selecting the whole equation, then choosing Multiply both sides, and then typing in 27 when prompted.

The necessity of entering arguments for operations means that the problems of one-dimensional versus two-dimensional notation can't be swept under the carpet entirely by providing problems on-disk. Fortunately, the arguments to

operations are typically quite short and simple. Moreover, they often (perhaps almost always) are expressions currently visible on the screen, and so the Term Selection Interface can be used. The remaining cases when somebody has to type in an argument should be very rare. I am unable to give an example.

7.4 Justifications

The operations must return not only the result of the calculation, but also a *justification* for display as the "reason" for the step. An interesting and subtle interface question arises here. On the one hand, these "justifications" must look right when printed in a solution, as a reason for a step. On the other hand, they should also correspond to what is written on the menu to invoke the corresponding operation, so that if one has a *Mathpert* solution in hand, one could in principle duplicate it. For example, if the student takes one step in auto mode, it should be apparent how that step could have been taken in menu mode. These criteria were not always easy to balance. For example, Factor out highest power might occur on a menu, and one might want the justification to be $ab + ac = a(b + c)$. But this justification won't lead the user to Factor out highest power. Careful attention to both desiderata while making up the justifications was necessary, and perhaps the best balance has not always been struck. With the introduction of the Term Selection Interface, this has become a less important issue, as the necessity to browse the menus has diminished considerably.

7.5 Interface to the grapher

Mathpert can make every kind of two-dimensional graph. It can make ordinary graphs of a function $y = f(x)$; it can graph two functions at once (on the same axes or on different axes); it can make polar or parametric plots. It can graph not only functions but arbitrary relations $f(x, y) = 0$. It can also show you the complex roots of a polynomial, or solve an ordinary differential equation.

It is one thing to make a grapher and another thing to make a grapher that is usable for classroom demonstrations and independent use by students. You must be able to get the graph you want quickly. You must be able to show the effect of changing a parameter *quickly*, without having to go through an elaborate data-entry system first. For example, we want to draw a series of graphs of $y = x^3 - ax$ for $a = 1, 2, 3, 4, \ldots$, and then $a = 1, 0, -1, -2, \ldots$.

You must be able to "zoom up" to a larger portion of the graph or "zoom down" (microscope style) at will, without data entry and waiting. You should be able to change every detail of the graph's appearance (line width, colors, etc.) yet not be distracted by screenfuls of such information when you don't want to change these things.

Some users reported to me that they find it important to be able to put a grid over the graph and remove it at will. People want to be able to make the

ticks on the x-axis occur at multiples of π when they are graphing trig functions.

The ability to change parameters and axis ranges quickly is vital to education. For example, given a grapher that can plot the complex roots of a polynomial, if you can change parameters rapidly you can do the following: show how the roots of $x^2 - bx + 1$ move when b is changed, at a speed fast enough that the movement appears to be animated. (They move around the unit circle, through a bifurcation where they meet at $x = 1$, then spread out in opposite directions on the x-axis.) One can also make n a parameter and see how the roots of $x^n - 1$ depend on n.

Small details of an interface often make a difference to the "look and feel" of software. An example that comes to mind is the question of putting titles on graphs. People want the title of the graph to use normal (two-dimensional) mathematical notation. That means it won't fit into a one-line title bar and should be placed on the graph itself. But then, the user must be able to move the title (with the mouse) to the location where it "looks best". This will vary from graph to graph. Moreover, if two graphs are being drawn on the same axes, the two titles need to be independently movable, so they can be placed near the corresponding curves.

Given the above requirements, what is the correct interface design? One wants to provide tools to accomplish the above tasks, in such a way that users will see and use the tools, but without cluttering up the screen, and adhering to the design principle of ease of use. Since there are so many things you can do to change your graph, it's not easy to provide a complete set of tools for accomplishing all those things *easily* without overwhelming the user with too many confusing icons, buttons, and menus. *Mathpert* tries to provide a small number of the most commonly-used tools in a visible form as a *Graph ToolBar*, occupying a narrow vertical strip to the left of each graph. The rest of the tools are accessed via menus. Using the ToolBar, you can zoom in or out and change parameter values with a single click.

One interesting issue in the design of a grapher interface is whether the graph should be instantly updated when a change is specified, or whether the user should be able to specify several changes before redrawing. The increasing speed of computers that run *Mathpert* has weighed in favor of instant redraw, for example when a zoom button or parameter-increment button has been pressed. Avitzur's Graphing Calculator (see Avitzur [1995]) sets a high standard for a grapher interface, and the Power Macintosh on which it runs certainly has enough speed to make instant redraw attractive. But there are a few circumstances where instant redraw is awkward, namely when the redraw is not actually instant, but takes too long. For example, looking at the partial sums of a series with the number of terms as a parameter, you will not want to change the number of terms from 5 to 50 by increments of 1 if you have to wait for each graph along the way to be drawn. (Such a formula can't be entered in the Graphing Calculator.) Another example is a contour plot. In most situations, however, instant redraw is preferable, so the cases mentioned have to be handled

as well as possible by other features of interface design, such as easily access to dialog boxes for changing the parameter value by large amounts, and a button to interrupt a slow drawing.

8 Interfaces and pedagogy

The development of software for symbolic computation has already caused substantial debate about the mathematics curriculum and pedagogy. The development of education-specific software for computations and graphs will have further effects.

Here is a single example of what can be done. When teaching Calculus II using *Mathpert*, one day I introduced exponential functions $y = a^x$ graphically, letting them change the parameter a. I asked each student to find the value of a which gives the graph slope 1 as it crosses the y-axis. They had to read the numerical value of the slope (which is easy with *Mathpert*), and adjust the value of a accordingly. In this way they each soon arrived at value of e accurate to several decimal places. Then we used the graphical ODE solver to look at the solutions of $y' = y$, and saw their similar shape. We then calculated the derivative of e^x symbolically, using the definition of the derivative as a limit. Finally they were given a file of homework problems on differentiation of functions involving exponentials.

I promised at the beginning of the paper to return to the issue of software that would not only replace pencil and paper, but also teachers and books. One preliminary step in this direction is afforded by the Microsoft Windows technology known as OLE (Object Linking and Embedding) [7], and by the competing Apple technology OpenDoc. This means that documents created by one software application (such as a spreadsheet or *Mathpert*) can be embedded in a document created by another application (such as a word processor). In practice it works like this: You write a page of a textbook in your favorite word processor explaining the exponential function $y = a^x$. You make a graph of it in *Mathpert* and paste that into your document using tools that today come with Windows. You include instructions for the student to click on the graph and find the value of a that makes the slope be 1 where it crosses the y-axis. The student reading your document clicks on the graph, and *Mathpert* starts up on that document. When the student has completed the exercise, she exits *Mathpert* and is back in your textbook, ready for the next part of the exercise, as described above. OLE technology means that anyone who can write textbooks can also write teaching materials that incorporate symbolic computation software. Publishers are nowadays busy putting existing textbooks on CD-ROM, with hypertext links from subject to subject. In the future, such CD-ROM based texts will also have live links to symbolic computation software. Recently there has been a wave of speculation that instead of CD-ROM, the base for hypertext educational material should be the World Wide Web.

Software to replace teachers and books will certainly be "multimedia", meaning it will incorporate sound and video. These features will have to be incorporated with full respect to the principle that the user is in control. Such software will not consist of videotaped 50-minute lectures. It *will* have live links to short video presentations, which might be either lecture-style (but limited to one short topic), or videos designed to show the applications of mathematics. Maybe you could click on the double-angle formulas for trigonometry and get a video lecture deriving them, and click again and get a video showing how they are used to design airplane wings or engines. A click in the right place might bring you a short history of those formulas, with biographical details if desired.

I have referred above to the curricular reformists who would have us emphasize concepts instead of techniques. I think this argument will fade away with the introduction of software that is capable of carrying out step-by-step solutions, thus supporting instruction in mathematical techniques. Would you like a mechanic working on your car who has been trained in the concepts of engine design? Well, of course, but she had better also know how to service the car. Similarly, we want the engineers who design jet planes to have a thorough background in mathematical techniques, as well as concepts. I don't think there is a controversy here. I believe the development of software meeting the design criteria set forth in this paper will enable the teaching of *both* techniques and concepts, in an integrated curriculum that has yet to be developed. Both the design and the delivery of this curriculum will be profoundly influenced by technology.

9 Use and availability of *Mathpert*

Development of *Mathpert* began in 1985; by 1989 I was using it to teach classes at San Jose State University, but the students had no access: it was used only for classroom demonstrations. In 1990 I was awarded an ILI grant from NSF to open a student mathematics learning lab at San Jose State, and taught two semesters of calculus to students in that lab using *Mathpert*. Several other faculty members also used *Mathpert* for a few classes in that laboratory. By 1994 the calculation engine of *Mathpert* was nearly finished, but the interface (DOS) was five years out of date. Therefore an interface was developed using Microsoft Windows 3.1 in 1994-95. As of this writing (September 1995), *Mathpert* is projected to reach the market in early 1996.

References

[1] Avitzur, R., Direct manipulation in a mathematics user interface, this volume.

[2] Avitzur, R., Bachmann, O., and Kajler, N., From honest to intelligent plotting, *Proceedings of the ACM International Symposium on Symbolic and Algebraic Computation (ISAAC 95), July 1995, Montreal, Canada*, ACM Press.

[3] Beeson, M., Logic and computation in Mathpert: an expert system for learning mathematics, in: Kaltofen, E., and Watt, S. M., *Computers and Mathematics*, pp. 202-214, Springer-Verlag (1989).

[4] Beeson, M., Mathpert: a computerized environment for learning algebra, trig, and calculus, J. Artificial Intelligence and Education 2 (1990), pp. 1-11.

[5] Beeson, M., *Mathpert*: Computer support for learning algebra, trigonometry, and calculus, in: A. Voronkov (ed.),*Logic Programming and Automated Reasoning*, Lecture Notes in Computer Science **624**, Springer-Verlag (1992).

[6] Beeson, M., Using nonstandard analysis to check the correctness of computations, to appear in *International Journal of Foundations of Computer Science*

[7] Brockschmidt, Kraig, *Inside OLE 2*, Microsoft Press, Redmond, Washington (1994).

[8] Buchberger, B., Should students learn integration rules?, ACM SIGSAM Bulletin **24** (1990) pp. 10-17.

[9] Bundy, Alan, *The Computer Modelling of Mathematical Reasoning*, Academic Press, London (1983).

[10] Burton, R. R., Diagnosing bugs in a simple procedural skill, in D. H. Sleeman and J. S. Brown (eds.), *Intelligent Tutoring Systems* pp. 157-185, London, Academic Press (1982).

[11] Clarke, E., and Zhao, X., Analytica–an experiment in combining theorem proving and symbolic manipulation, Technical Report CMU-CS-92-1 7, School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213.

[12] Fateman, R., Hones plotting, global extrema, and interval arithmetic, in P. S. Wang (ed.), *Proceedings of the ACM International Symposium on Symbolic and Algebraic Computation (ISAAC 92), July 1992, Berkeley, USA*, pp. 216-223, ACM Press.

[13] Montessori, Maria, *The Absorbent Mind*, Holt, Rinehart, and Winston, New York (1967). Translation of the Italian original published in 1949.

[14] Nguyen-Xuan, F., Nicaud, J. F., Gelis, J. M., and Joly, F., Automatic diagnosis of the student's knowledge state in the learning of algebraic problem solving, *Proc. Artificial Intelligence in Education*, Edinburgh (1993).

[15] Nicaud, J. F. et. al., APLUSIX: a learning environment for acquiring problem solving abilities, Proc. COGNITIVA '90, Madrid (1990).

[16] Nicaud, J. F., Gelis, J. M., Saidi, M., A framework for learning polynomial factoring with new technologies. International Conference on Computers in Education 93, Taiwan (1993).

[17] Ravaglia, R., Alper, T., Rozenfeld, M., and Suppes, P., Successful pedagogical applications of symbolic computation, this volume.

[18] Richardson, D., Some unsolvable problems involving elementary functions of a real variable, J. Symbolic Logic 33 (1968) 511-520.

[19] Wu Wen-Tsun, Basic principles of mechanical theorem-proving in elementary geometries, *J. Autom. Reas.* **2** (1986) 221-252.