N. Kajler (Ed.), Computer-Human Interaction in Symbolic Computation,
Springer-Verlag, 1999, pp. 1-29.

Successful pedagogical applications of symbolic computation

Raymond Ravaglia, Theodore Alper, Marianna Rozenfeld, and Patrick Suppes

1 Introduction

At the Education Program for Gifted Youth (EPGY) we have developed a series of stand alone, multi-media computer-based courses designed to teach advanced students mathematics at the secondary school and college level. The EPGY course software has been designed to be used in those settings where a regular class cannot be offered, either because of an insufficient number of students to take the course or the absence of a qualified instructor to teach the course. In this way it differs from traditional applications of computers in education, most of which are intended to be used primarily as supplements and in conjunction with a human teacher.

Since the Fall of 1990 EPGY has developed a body of computer-based courses designed to teach mathematics from the Kindergarten level through the first two years of university level instruction. Students participating in EPGY enroll formally in these courses through the Stanford University Continuing Studies Program and receive credit for the courses they complete. To date we have had over six hundred students complete courses with us, with about one quarter of those in university level courses. We expect the total enrollment in EPGY for the Fall Quarter of 1995 to exceed one thousand students. (For a detailed discussion of our early results see Ravaglia, Suppes, Stillinger, and Alper 1995.)

In reading this article it is essential to keep in mind that our software is not something which has remained in a lab being poked at by computer scientists or mathematicians, only occasionally to be used by students. Rather students are using it regularly, and as such much of our development has been motivated by their day to day needs. Theoretical concerns have often had to assume a secondary role. While this has been frustrating at times, it has also forced us to be extremely honest in evaluating what aspects of our courses play a significant pedagogical role and what aspects are extraneous.

In designing these courses we have tried to be as responsive as possible to variations in student abilities and rates of learning. Because of the opportunities for assessment it affords, symbolic computation has been an essential tool in the instructional design of these courses. In this paper we will briefly examine the standard ways in which symbolic computation is incorporated into mathematics instruction. We will focus not on the ben-

efits that such an incorporation affords, but rather on how specific features common to symbolic computation programs diminish their pedagogical effectiveness.

These shortcomings will be addressed in the context of a discussion of the EPGY course software in which we will illustrate how we have incorporated symbolic computation into the program and what pedagogical effects it has had. In particular, we will describe how existing symbolic computation systems suitably modified and coupled with a semantic component, make possible systems in which students can construct derivations and receive immediate feedback. Such derivation systems make it possible to isolate and develop computational skills, without resorting to standard drill and practice techniques. Furthermore, derivation systems, by forcing students to justify explicitly their inferences, make it possible to evaluate student's understanding of a given body of mathematics. This in turn makes it possible to assess students at a deeper level than is usually done in computer-based courses. Ultimately it is the individualized adaptive nature of instruction afforded by the ability to perform this type of assessment which makes it possible to develop successful computer-based courses.

2 Symbolic computation and pedagogy

The last half dozen years have seen the development of a number of symbolic computation packages available on personal computers. Coincident with the development of these packages have been numerous attempts by educators, computer scientists and mathematicians to incorporate these powerful tools into secondary school and college level mathematics curricula. A sense of the shear number of these attempts is nicely illustrated by looking through the presentation list of the Seventh Annual International Conference on Technology in Collegiate Mathematics (ICTCM 1995). Over a third of the presentations involved discussions of how symbolic computation programs have been used to develop instructional material for standard mathematics courses.

The common thrust of the bulk of these efforts has been to provide students with what is essentially a mathematics laboratory environment in which they can explore the properties of certain mathematical objects without having to do any computations themselves.

Depending on the pedagogical goals underlying it, this instructional approach takes several different forms. In its weakest form, students are presented with an unstructured environment in which they are encouraged to experiment freely with some given mathematical object. For instance, students might be given a certain function and told to search for interesting properties that it might have. In the strongest form this approach leads to programs in which the exposition of the curriculum and the symbolic computation package will be linked together. In these cases students will be required to investigate specific properties of mathematical objects (graphs, tables or sequences of calculations) within the symbolic computation environment in order to complete an assignment.

Within this model of mathematical experimentation, different features of symbolic computation programs are emphasized. The most common is to utilize the graphing capabilities of symbolic computation packages to encourage students to think visually about functions. Perhaps the finest tool developed towards this end is the Graphing Calculator developed by Ron Avitzur (see his contribution in this volume for a thorough

discussion of this tool). The influence of this graphical approach is also present in the efforts of calculus revisionists to incorporate graphing calculators into calculus instruction. (A development of calculus which thoroughly incorporates the use of graphing calculators is Hughes-Hallett et al. 1992.) That the merits of this approach are widely accepted is evidenced by the decision of the College Board, a national organization which sets standards on how to teach college level courses in secondary schools, to require the use of graphing calculators on the national Advanced Placement Examination in Calculus beginning with the 1994-95 exam year.

The other feature of symbolic computation programs most commonly used is their ability to do infinite precision numerical computations. Such computations are frequently used to illustrate points about functions and limiting processes. By actually evaluating functions at points very close to the limit value, one can obtain very nice numerical tables demonstrating limiting behavior. Similarly one can demonstrate facts about derivatives by computing difference quotients at various points close to the point in question. Because of the ease and rapidity with which one can do these numerical computations, symbolic computation programs make it feasible to incorporate such examples into the curriculum.

A project which thoroughly incorporates these features of symbolic computation programs into mathematics instructions is the Calculus and Mathematica Project at the University of Illinois (Brown, Porta, and Uhl 1991 – for other suggestions see Crandall 1989 or Wagon 1991). Uhl and his colleagues have abandoned the traditional lecture style of teaching calculus for an interactive laboratory setting in which students use Mathematica to work through a series of problems and examples combined with instructional text.

We will not discuss the merits of these approaches here as they are addressed extensively in the literature (again see Brown, Porta, and Uhl 1991, Crandall 1989 & Wagon 1991). Instead, we wish to focus on specific deficiencies of standard symbolic computation packages when used as pedagogical tools.

2.1 Either too much...

The greatest weaknesses of standard symbolic computation programs as pedagogical tools paradoxically stems from their intrinsic power and generality. In their recent book on computer algebra Davenport, Siret, and Tournier (1988) state that computer algebra systems as normally conceived are expected to meet two requirements:

1. Provide pre-programmed commands to perform wearisome calculations.
2. Provide a programming language to define extensions or enlargements of the original set of pre-programmed commands.

Neither of these desiderata is likely to generate symbolic computation systems well suited for educational purposes.

The first requirement is problematic because such pre-programmed commands which perform all the “wearisome calculations” obscure many of the details which a student needs to see and learn as part of the process of learning mathematics. Using one command to integrate, for example, ignores the importance such techniques as integration by

parts, or integration by trigonometric substitution. The importance of these techniques transcends the merely computational, and teaching them comprises a good deal of the third quarter of calculus. While the Maple "simplify" command might embody all the knowledge of a second year algebra course, teaching someone how to apply this command in Maple is very different from teaching him second year algebra.

Also obscured here are many of the subtleties involved in performing certain calculations. The standard symbolic computation programs do not worry about this sort of error checking since they are designed for use by scientists who already know the mathematics, and can detect erroneous results stemming for incorrect applications of rules and ignore them. This is not an assumption which should be made about someone just learning the material. What is desirable for the expert is often harmful to the student.

The second point is equally problematic. Because programs like Maple and Mathematica have been designed for general use by scientists and engineers, certain assumptions have been made concerning the user interface and the expected power of the system. Scientists and engineers can be assumed to be well versed in programming languages. For this reason, command language interfaces are the norm for symbolic computation programs. This is understandable since adopting a command language similar to a programming language enables one to create a symbolic computation environment which is readily extendable.

Students, however, cannot be expected to master anything this complex. While it is reasonable to expect students to use calculators in mathematics courses since the interface of a calculator is almost transparent, it is not *a priori* reasonable to expect students to learn the interface required to use a symbolic computation program. Moreover, even if being proficient with the use of symbolic computation programs correlates with an ability to learn calculus, this fact does not justify the introduction of symbolic computation. A similar claim might be made concerning proficiency with first order logic. It is probably the case that a student who knows first order logic is not subject, when learning calculus, to the same sorts of confusions concerning quantifiers in proofs using $\epsilon - \delta$ methods as someone who has not had logic. It does not follow, however, that students should be required to take a course in logic before they take a course in calculus. For this to happen one must first establish that the students who have difficulties with these concepts in the calculus will have an easier time learning them in a logic course. And even then one must show that the time spent first learning logic might not have been more productively spent otherwise.

2.2 ...or too little

The single greatest advantage that computers have as an instructional tool is their ability to individualize instruction to meet the needs of a particular student. This principle, recognized since the earliest drill and practice mathematics programs, remains the essential advantage of computer instruction, for while a teacher lecturing must address an entire class at once, a computer, by assessing each student's understanding, and tracking that student's performance over time, can adjust the level of instruction to match that particular student's needs.

Ideally both the amount of material presented to a given student and the level at

which it is presented should vary according to the rate at which the student masters the material. Students who can move quickly through a given subject should be allowed to, while those who need additional instruction or a more concrete style of presentation should have those resources available to them. To accomplish this within a computer-based course requires assessment of a student's understanding at each point in the course, together with a record of that student's performance through the course up to that point. The judicious use of symbolic computation makes this type of assessment possible.

It is surprising therefore that it is exactly this type of assessment which is missing in the standard incorporation of symbolic computation into the curriculum. The reasons for this lie in the limitations of the "Mathematica Notebook" approach to course development.

In the notebook model students are presented with what essentially are dynamically unfolding objects. Students can explore these objects and interact with them in real time. One can click on the equation of a function and produce its graph, rotate it in three space, and automatically evaluate its integral. Or one can investigate a function's limits by creating a table of values with arbitrarily great decimal precision. While such experimentation has its place in a computer-based mathematics course, it is not sufficient in itself to constitute a course.

The biggest problem with those who decide to develop courses within the confines of the notebook model is that it is fundamentally a static, non-adaptive model since the notebook cannot modify the presentation of material to reflect the results of student assessment. Consequently, the interaction the students have with these notebooks occurs independently of what their degree of mathematical sophistication is. This is not only a poor utilization of a powerful resource, it is a poor approach to instruction.

3 The EPGY course software

The EPGY course sequence in mathematics consists of on-line courses which cover the standard curriculum in the following courses: First Year Algebra, Second Year Algebra, Precalculus (includes Trigonometry and Analytic Geometry), Differential Calculus, Integral Calculus, Multivariate Calculus, Linear Algebra and Differential Equations. Each of these courses is designed to be complete and comparable to what a student would receive in a standard high school or college level course. In this paper we will focus on the sequence from Beginning Algebra through the Integral Calculus.

The EPGY courses are completely computer-based. They are distributed on two or more CD Roms and are designed to run locally. Each course consists of lessons which correspond to the logical sections in a text book. Lessons begin with a multi-media presentation in which digitized sound is played and synchronized in real-time with the display of graphics to create what resembles a teaching writing on a blackboard while lecturing. Students have full control over the lectures being able to pause, fast-forward and rewind at any time. Students can control several lecture parameters including the speed of the lecturer's voice and the format of the graphic display (see Figures 1 and 2).

It is worth noting that these lectures have been designed so as to preserve the informal nature of spoken mathematics or physics as contrasted with the more formal prose

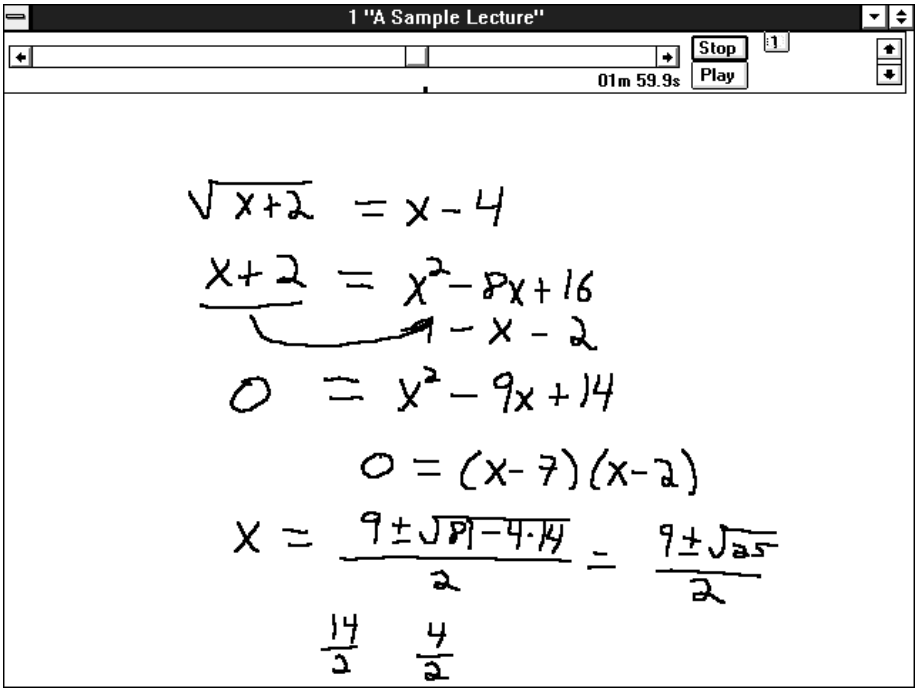


Fig. 1. Handwritten Algebra Lecture

style of text books in these subjects. This is important since it has been observed by many people, though never adequately researched, that oral lectures are an important part of learning the mathematical and physical sciences. We believe that such informal lectures are important; they allow the students to absorb matters of style, such as how to talk informally about the subject, how to draw diagrams, and how to write equations.

The lectures are followed by a set of simple questions which review the students' understanding of the material just presented. After these review questions students are presented with a set of interactive exercises, which consist either of a quiz on the material covered in the lecture, interactive exposition in which the student is lead through a detailed argument step by step, or a derivation in which the student is asked to obtain the answer to an exercise. As one would expect, the difficulty level of the exercises increases as the student progresses into a lesson. Depending on its complexity the student may have to make several intermediary computations. These computations can be done either with paper and pencil or by using our Derivation System.

In addition to the CD Rom based instruction students have interaction with remote human instructors and each other through a variety of electronic means. As these features are not directly relevant to the topic at hand, and as they have been discussed elsewhere (Ravaglia, Suppes, Stillinger, and Alper 1995; Ravaglia, Barros, and Suppes 1994),

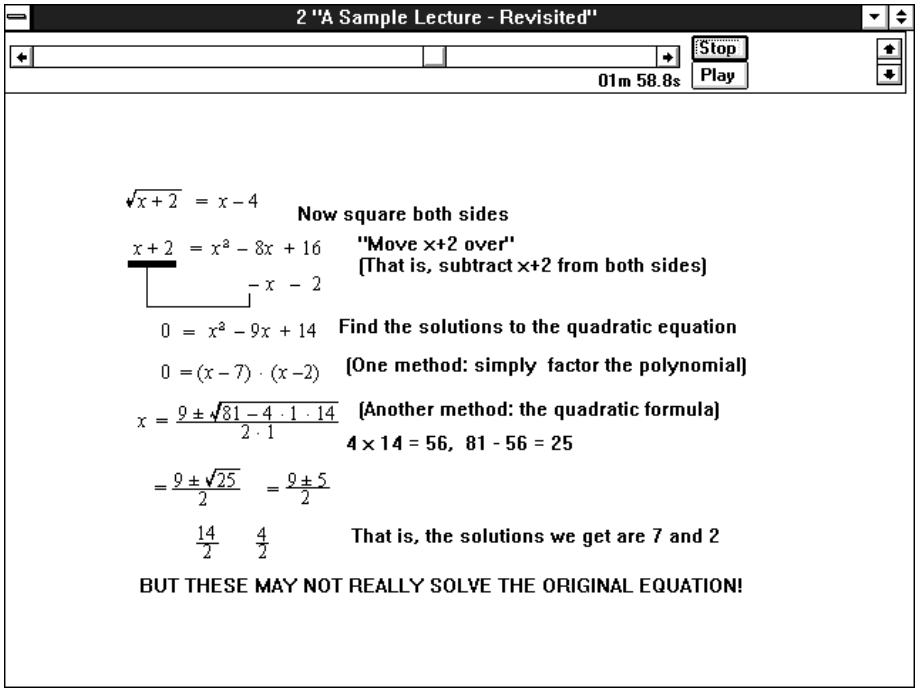


Fig. 2. Formatted Algebra Lecture

we will not discuss them here.

3.1 Student assessment & symbolic computation

To be responsive to a student's level of understanding, a computer-based course must be able to assess the student's comprehension of the material and adapt itself accordingly. Students demonstrating ready mastery should be allowed to move quickly through the material while students who are having difficulty should be given appropriate remediation. Ideally this sort of assessment must determine two things. The first is whether or not the student is able to produce the correct answer to the sorts of questions that the student will see on examinations. The second is whether or not the student who produces correct answers actually understands why the answers are correct.

Assessing whether or not students can produce correct answers is relatively straightforward and can be done by asking students the sorts of questions that instructors traditionally ask after lectures or on examinations. However, in designing such free answer questions several issues must be taken into consideration.

The most basic is the ease of input. If a student has to type a complex mathematical expression in an input language, the chance that an incorrect response is caused by an

error in typing rather than an error in understanding will be such as to greatly diminish the assessment value of the question. Care must be taken to provide the student with both a convenient means of input such as typesetting programs do, together with the ability to see their input formatted, so that they can verify that what the computer has understood is in fact what they wished to express (see Figure 3).

Fig. 3. Structural Input System.

Equally important with ease of input is the naturalness of input. Students should not have to constrain their answers to fit a particular form, outside of those constraints which an instructor would reasonably place upon them in class. While it is reasonable to require fractions to be expressed in lowest terms, it is not reasonable to prefer prime notation to Leibniz notation. Students must be allowed to concentrate on getting the problem correct without worrying about expressing it in a form that the computer can understand.

As such it is not surprising that inflexible processing of student answers is a standard complaint with computer taught courses. Because answer comparison is most often some variation of string matching, students are required to follow rigid input conventions. Unfortunately this eliminates the wide variety of natural variations in the ways people express even relatively simple mathematical expressions.

Our approach has been to process the answers symbolicly, taking into consideration their mathematical meaning, and considering possible correct answers in terms of equivalence classes. This allows the computer to understand natural variations in correct answers. Moreover, since these variations often arise as the result of different approaches to the solution of a problem, an improved ability to accept variations in answers results in a direct increase in the flexibility that students are afforded in choosing how to solve problems.

A simple example from algebra shows the natural variety that a student's answer can take. Suppose a student is asked to solve the equation $x^2 + x + 1 = 0$ in the complex plane. One may want to accept as correct all of the following variants: $\frac{-1+i\sqrt{3}}{2}$ and $\frac{-1-i\sqrt{3}}{2}$; $-\frac{1-i\sqrt{3}}{2}$ and $-\frac{1+i\sqrt{3}}{2}$; $\frac{-1}{2}+\frac{i\sqrt{3}}{2}$ and $\frac{-1}{2}-\frac{i\sqrt{3}}{2}$; $\frac{-1}{2}+i\frac{\sqrt{3}}{2}$ and $\frac{-1}{2}-i\frac{\sqrt{3}}{2}$, not to mention several others with essentially the same form. To code each of these pairs of answers for the purposes of simple string comparison would be a chore and would fail

to exploit the semantic content of the mathematical expressions.

Whether or not the student's answer is correct can be determined by passing the student's input and the author coded answer plus specification of the equivalence class to a symbolic computation program for evaluation and comparison. Exploiting the fact that the answers are mathematical expressions increases the flexibility for student input and simplifies author coding.

The importance of the author specifying an equivalence class in addition to an answer must be stressed. If full blown equivalence is tested for between author and student answers, problems in which the student was supposed to have transformed an expression according to algebraic rules into a new expression would be useless. This is because the student could just type in the initial expression, which in the case of problems from an algebra course, is almost always algebraicly equivalent to the answer. This is another example of where the goals of symbolic computation programs and human instructors diverge.

A good example of this problem are exercises on factoring or multiplying polynomials. The initial expression is algebraically equivalent to the final one, as are all the intermediate stages of the computation, but there is still a definite correct answer. Also, in expressing the factorization of a polynomial, one does not care about differences in order of the terms, or in terms of whether a factor of negative one has been introduced, but one does care that the expression has been factored and as such, one will not extract factors from the student's answer or multiply out the author's. Thus if the problem is to factor $x^2 - 5x + 6$ one would accept as correct $(x-2)(x-3)$ or $(x-3)(x-2)$ or $(2-x)(3-x)$, and so on, but not $x(x-5)+6$.

The size of the set of answer variants which one will accept as correct will depend on the course the student is in as well as the material recently covered in that course. For example a student in beginning algebra, particularly one just learning that i stands for $\sqrt{-1}$, might be required to resolve all negative roots into expressions containing i. In this context $\frac{-1+\sqrt{-3}}{2}$ would not be accepted as equivalent to $\frac{-1+i\sqrt{3}}{2}$, while $-\frac{1-i\sqrt{3}}{2}$ would be. Contrasted to this, an answer from a student in the calculus, just learning derivative rules might be considered correct as long as it was algebraicly equivalent to the author coded answer and contained no unevaluated functionals.

3.1.1 The EPGY derivation system

We have just seen how the judicious use of symbolic computation to evaluate answers plays an important role in assessing the ability of students to solve problems. What this use of symbolic computation does not provide in this context, however, is a way to look at the process a student goes through in solving a problem. While getting the correct answer is the goal of solving a problem, getting it in the correct way is equally important. Hence the mathematics teacher's dictum "Show your work."

The EPGY software has addressed this issue by creating a derivation system, i.e. an environment in which students can formally manipulate mathematical expressions by applying inference rules. In the standard environment, the student supplies the rule and the derivation system takes care of performing the appropriate calculation. The results of the calculation are preserved for the student to further manipulate. A derivation of a

problem is the set of steps from the statement of the problem to the solution. In addition to the regular exercises students work in the course they are required to work a certain number in the derivation system.

A derivation system as we conceive of it differs from a raw symbolic computation environment in two primary ways. The first is in having the logical structure necessary to represent mathematical inference and logical dependency. The second is in providing students with a set of rules to use which are appropriate to both their level of understanding and position within the course.

The importance of the first point can be quickly illustrated by a trivial example of a fallacious inference permissible in most symbolic computation systems.

```
Assume: a = 0.
Divide both sides by a: a/a = 0/a.
Simplify: 1 = 0.
```

If one conceives of the intermediate results of one's use of a symbolic computation system as being unrelated, one cannot discuss inference. Because most symbolic computation systems are devoid of a semantic component, they impose no structure on the sequence of steps which a student produces in working a computation, and as such, allow students to produce obvious contradictions. What is needed for students is a system which treats the results as a sequence of related lines moving from an initial assumption via accepted inference rules to a valid conclusion, as one does when giving a proof.

The importance of the second point, the need to provide students with rules appropriate to their level of understanding, is one which most symbolic comptutation environments have chosen not to pay attention to. This choice is understandable given the population that such symbolic computation programs are designed for, especially since to do so is essentially to reduce the power of the system. However, while it is understandable, the choice is damaging from a pedagogical perspective. While it may be the case that a physicist using a symbolic computation package to solve a computationally tedious problem does not need to worry about inferences in moving step to step to the solution, most students learning the material for the first time are not so sure footed. The major difference between students and experts here lies in the assumption that experts can refine the performance of the software with their own knowledge. Experts can be expected to throw out obviously bad results or massage them into a correct form as necessary. It cannot be assumed that students are able to make these same corrections.

Our contention is that, rather than providing powerful commands and extendability, what a symbolic computation system must provide to properly function as a pedagogically useful derivation systems is a way to represent mathematical inference and logical dependency.

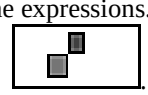
4 User interface & design issues

The EPGY derivation system is present in our sequence of courses from the first year of secondary school algebra through first year of college calculus. In designing this system we have tried to make its interface as transparent as possible so as to minimize the amount of time students must spend learning its use and maximize the probability that

the incorrect application of a rule is due to a conceptual mistake and not a consequence of bad interface design. We have incorporated the structural input system described above into the derivation system interface as well. We have also been motivated by the desire to make it such that all transformations which can be made correspond to an explicit application of a rule. This is essential if every inference is to be explicitly identifiable. We have also required that the system be sensitive to the differences in ability and levels of understanding that the different students using it will have. A feature essential for a calculus student may not always be suitable or even desirable for a student just learning algebra.

In our design we have eschewed both command language interfaces and contextually driven menu interfaces for a number of reasons. Command language interfaces, the norm in commercial symbolic computation packages like Mathematica or Maple, require class time be spent learning and mastering their syntax, knowledge which may be useless outside of class. Menu interfaces, on the other hand, are more easily mastered, but may be cumbersome to use. If one does not impose a context based bias on the selection, the user has to wade through a long list of rule possibilities and select the one he or she wants. If one does impose a context based bias, then the resulting derivations cannot be said to be natural.

What we have settled on for the rules is a set of nested palettes with icons representing the rules. Both the rule groups and the rules themselves are represented by graphical icons depicting the form of the group or rule. To apply a rule, a student first selects the expression or subexpression to which the rule is to be applied, enters any required parameters for the rule into the input window, and then selects the appropriate rule from the rule window. These palattes differ from menus in having all commands immediately ready to hand, rather than burried in linear lists. Moreover palettes naturally lend themselves to the use of icons which in turn exploits the density of information afforded by images.

The icons themselves graphically represent the rules for which they stand, but avoid (in general) the direct use of standard algebraic notation, relying instead on abstract shapes to represent variables or terms within the expressions. So, for example, the exponential rule group is designated by the icon:  . Selecting the rule group causes the specific rules for that group to appear in the palette to the immediate right of the rule group palette (see Figure 5).

In the derivation system one can use the mouse to directly select the subexpressions to which one wishes to apply a certain rule. While we have allowed certain keys to cause transformations to the expression being worked on, we have not followed in the direction of Avitzur (see his contribution in this volume) in interpretting mouse movements as commands. The reason for this is the desire to require that students be explicit in their intent to apply rules and in expressing exactly which rule they wish to apply. We believe that exactitude on these issues is important for adequate assessment.

The use of squares with different colors to stand for expressions in the icon, rather than letters, simplifies the process of binding variables when applying rules. This is because the student does not have to suffer the potential confusion caused by having a variable name used in two different ways, such as when a student attempts to apply a rule

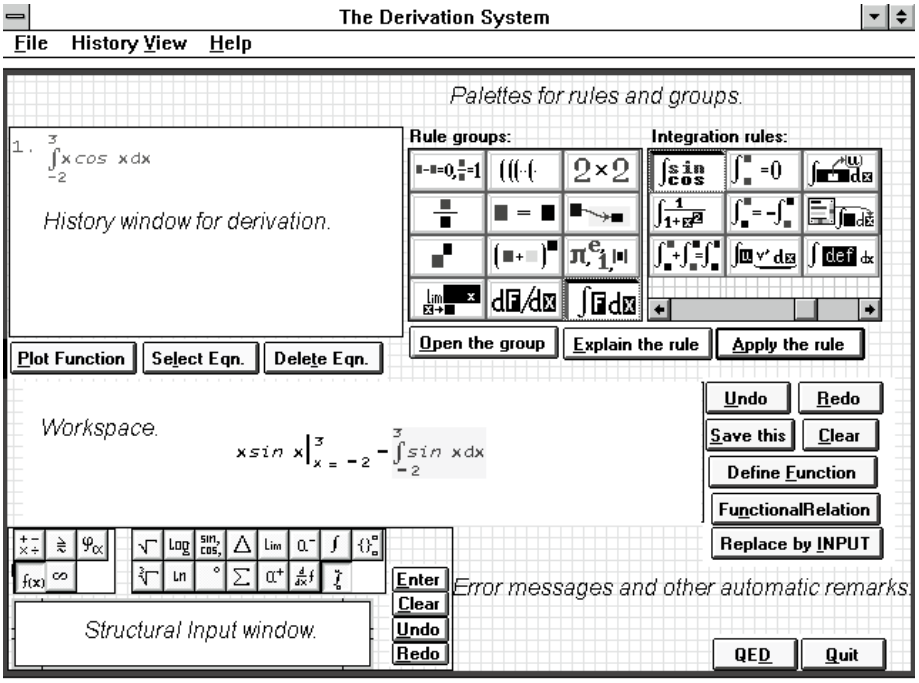


Fig. 4. The Derivation System.

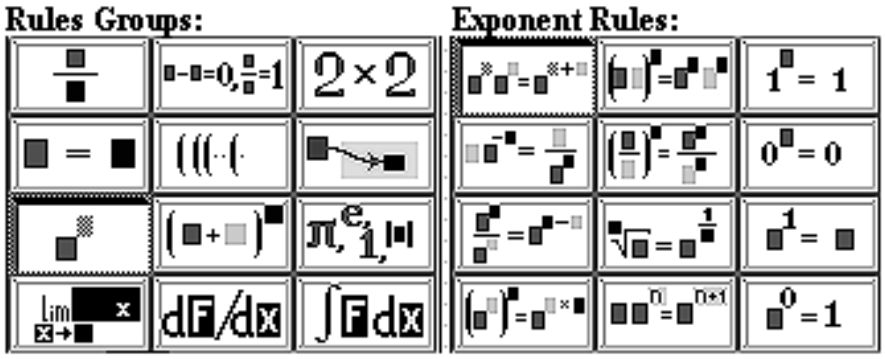


Fig. 5. The Rules Groups and the Exponent Rules.

Fig. 6. Adding Fractions with and without Common Denominator.

such as:

$$x + y \rightarrow y + x$$

to the expression $y + x$. Requiring the student to specify that y is the value for x and x for y in the statement of the rule can cause confusion (see Suppes and Sheehan 1981).

In the derivation system, the rules are organized hierarchically within rule groups, according to the type of rule they are. Exponential rules are grouped together, as are rules for grouping, limit rules, differential rules, integral rules, and so on. For convenience we do not adhere to strict categorization, but allow the same rule to belong to several groups. Thus the rule which says that exponentiation by unity leaves an expression unchanged is placed in both the group of exponentiation rules, and in the group of rules reserved for arithmetic simplifications.

This sort of arrangement allows students to find quickly the rule they need to perform a given calculation. If a student wishes to add two fractions, the student first selects the fraction rule, and then determines if the fractions have a common denominator, and then applies the appropriate rule (see Figure 6).

Naturalness of rules

The rules in the derivation system have been designed to correspond to the theorems one learns in these courses. We have augmented these rules with others which are seemingly redundant, but which facilitate doing certain types of computation. We have tried to make the student's approach to solving problems within the derivation system parallel to the approach the student would take solving the problem with pencil and paper. We have hoped in this way to make work done in the derivation system help the students to develop skills which will carry over to their own work.

Integrated of exposition and derivations

We have integrated the derivation system into the development of the standard secondary school curriculum from the beginning. As soon as students learn an algebraic fact in the course they are taught a corresponding rule in the derivation system. In fact the derivation system has been constructed in such a way that the rules are not present in the system until after the student has formally learned them. In this way the state of the derivation system for a student at a particular time reflects what that student has learned up until that point.

Power of rules appropriate to student understanding

It should be noted that as the students' knowledge increases the power of the rules do as well. Certain computations that should be made explicit by students just learning algebra, would be needlessly tedious to do explicitly by a student in the calculus. The derivation system has been made sensitive to these differences and the rules have been designed to become more automatic in their handling of algebra the further along the student is in the course.

Students must demonstrate comprehension of rules

Every time a rule is introduced into the derivation system, students are expected to spend 5 to 10 minutes working simple problems to familiarize themselves with that rule. Afterwards they work a series of more difficult problems which require use of that rule. In addition to problems which must be solved within the derivation system, students have limited access to the derivation system when working other problems and may choose to solve some of them with the aid of the system.

There is an issue which arises here, however, concerning the nature of solving problems within the derivation system. As in any symbolic computation environment, the work that the student actually does is rule application. The student does not have to do the computations. As such, just because a student is able to arrive at the correct answer in the correct way, does not guarantee that the student completely understands the process. For instance, a student might know that to find the derivative of an expression, he must first differentiate, then apply derivative rules, and finally collect terms and rewrite the expression, without actually being able to perform the computations on pen and paper needed to arrive at the answer.

We achieve this within the derivation system by requiring students to earn the right to use a rule by demonstrating that they are capable of producing the correct answer themselves. When a student is first learning how to use a rule, he or she is often asked to supply the result after he or she has attempted to apply the rule. The system displays a partial form of the result and asks the student to supply the missing part. As the student becomes familiar with the rule, we decrease the probability that the student will be asked to supply the results of the application. By forcing a student to do this correctly a certain number of times at first, and occasionally throughout the course, we make sure that the student can do the computation embodied in the rule. Moreover by randomly forcing students to do this throughout the course we can make sure that students do not forget how to do these computations.

Metacognition

Having students articulate their reasons for the steps they take in a derivation not only allows for an evaluation of their understanding, it actually deepens it. Each valid rule of computation must be explicitly acknowledged as it is used. In solving a problem students may consider the power and applicability of several rules before choosing the one they consider most appropriate. Through the repetition of this process throughout a course students' comprehension of material is improved, and they are made more likely

to retain what they have learned (Cuasay 1992).

5 Logical structure of a derivation

Because the derivation system is supposed to represent inference, the lines in the derivation must be more than a collection of unrelated statements. Some sort of logical structure must be imposed. This structure needs to be more than chain of implication. Consider for instance the solutions to the equation $|x-3|=5$. In solving this problem, one first assumes that $x-3$ is positive and later that $x-3$ is negative, and solves the resulting equations in each case. Related rates and relative extrema problems afford even more dramatic examples. Even though we are making several assumptions, we are not interested in each of these assumptions being true at once, but only in what each of these assumption taken in turn allows us to conclude.

We need to make sure that the conclusion arrived at in each strand is correct, but do not need to worry about the compatibility of the different strands, unless we try to combine them in some way. When a rule is applied which causes two strands to be combined, one must verify that there are no contradictory assumptions in the strands. If there are none, the rule can be applied and the new step added. If there is one, then the rule is not applied and the student is warned that there is a problem with applying that rule. As such, the structure of the derivation is graph like and not simply linear. Each assumption creates a new branch of the graph, and each conclusion is a node, dependent on the assumptions from which it stemmed.

Intermediate steps in derivations and important partial results are stored on numbered lines in the derivation history. Because of space considerations and custom we write each assumptions on its own line, rather than trying to preserve the true tree structure of the derivation. When a given line is selected a system of color codes is used to indicate how the expressions in the derivation depend on each other. Students who wish to may select to see the entire dependency tree of the derivation. We do not display it all the time because we prefer to keep the immediate focus on the mathematics at hand by minimizing the logical machinery that the students need to be aware of.

5.1 Restrictions and logical machinery

For a derivation system to represent inference it is essential that the system be able to keep track of the assumptions that must be made about a given expression before one can apply a particular inference rule to it. These assumptions arise from restrictions on the application of the standard algebraic rules. We return to our example from above.

We started with the assumption:

$$1. \quad a = 0$$

and attempted to divide both sides by a. The particular inference rule we wished to apply in this case is the conditional statement: if one divides both sides of an equation by the same term, the result will be an equality provided that the term was not equal to 0.

If we attempt to apply this rule to line 1 using the term a, we can recognize that we have a contradiction, and as such can inform the student of this by displaying an

appropriate error message. Thus rather than adding the new line:

$$2. \qquad \frac{a}{a} = \frac{0}{a}$$

the derivation system displays the message "This rule is not applicable since it involves division by zero."

Ideally, one would require students to demonstrate that the antecedent of the rule is satisfied before one would allow them to apply the rule. This is not always practical, however, since one does not always have an explicit value for a given variable.

For this reason we preserve validity by adding the appropriate assumption to the derivation in the form of a restriction. For instance if we had:

$$1. \qquad a = 0$$

and wanted to divide both sides of the equation by some new term b we would add the lines:

$$2. \qquad b \neq 0$$

$$3. \qquad \frac{a}{b} = \frac{0}{b}$$

The reason we can add these two statements as separate lines rather than as just a single line:

$$2'. \qquad \text{If } b \neq 0, \text{ then } \frac{a}{b} = \frac{0}{b}$$

is because we are interpreting the relation between the lines in the same branch of a derivation as being linked by implication and as such there is no difference between the derivation with lines 1–3 and the derivation with lines 1 and 2'. Because line 3 comes from lines 1 and 2, any inference rule we apply to 3 must not have antecedents which contradict 1 or 2.

We call these lines added to the derivation because of their role in the antecedents of inference rules "restrictions." It should be noted that if we were to leave the conditionals intact, the resulting system would be needlessly more complicated in that it would require that the students make the assumptions in the antecedent explicitly and then apply modus ponens. By having the program add the antecedents automatically to the derivation as restrictions, it makes the resulting system much more suitable for students.

5.2 Additional restrictions

The restrictions discussed above are generated by the algebraic form of the expression in question and can be expressed in a natural way in equational form. This makes it natural to add these restrictions to the derivation system automatically. However, there are three other types of restrictions which also must be accounted for if the derivation system is to adequately represent inference.

The first class of restrictions are assumptions about differentiability and continuity. These assumptions can not always be translated into equational form. In these cases, even if the derivation system cannot establish the truth of an antecedent assumption, the

program should still add the antecedent assumption explicitly to the derivation. Thus if in the case of differentiating $f(x) = a^x$, the program could not determine whether or not f was differentiable, it should apply the rule and then add the assumption of differentiability explicitly to the history of the derivation. If an apparent contradiction is obtained at some later point, then this assumption may be identified as the source of the difficulty.

The second class of restrictions are assumptions concerning the domain over which a stated equality holds. Suppose one has the equations, $a = 3$ and $a = 5$. If these statements are supposed to hold at a particular point, then we may not allow some step in a derivation to rely upon both. If a is a function which assumes these values at different points, then internally we assign a an independent variable and track these as $a(t_0) = 3$ and $a(t_1) = 5$ to indicate that a attains these value at distinct locations. This assumption will also determine whether a can be considered as a function in terms of t.

The third class of restrictions are those concerning relations of dependency between variables. Often times in the calculus one makes decisions about which variables are independent and which are dependent. These relations determine the applicability of rules such as the implicit function theorem. Similarly, one's ability to convert equations that contain no explicit function notation into statements about functions in which the dependent variable is explicit depends on the assumptions one has made in the course of the derivation. For example, if one has a derivation containing the lines:

$$p. \quad x + y = 3$$

$$r. \quad 3x + y = 5$$

one can combine these equations and solve for x and y if one wants. However, one cannot do this if one has already made assumptions about the dependency relations between x and y. If one has previously decided that in both cases y is to be considered a function of x then the result of combining these equations changes the nature of the dependency, as x is not longer a free variable.

5.3 The need for restrictions

Unless one keeps track of the restrictions as they arise in the derivation system, the system will not meet the needs of students. If the result of a derivation is literally false, than students who do not catch the error will be misled and those who do catch it will be confused. Neither case is desirable.

The calculus is littered with examples where the failure of standard symbolic algebra packages to track such restrictions allow students to wander into nonsense. For instance, if one has the function

$$f(x) = a^x$$

and asks for its derivative, the standard response will be

$$f'(x) = a^x \log a$$

What the program does not note is that depending on the value of a there are restrictions that must be placed on the domain of the function. Moreover, the domain of the derivative is even more restricted.

Mathematically knowledgeable users of the software understand these domain restrictions and know how to avoid getting into trouble. A student is in no position to do this, particularly if the software itself gives no warning that there is anything unusual about the result. The problem is that the symbolic package has applied the derivative rule for expressions of the form a^x, *viz.*, $\frac{d}{dx}a^x = a^x \log a$, without examining issues of domain. A student might conclude from this that $\frac{d}{dx}(-2)^x = (-2)^x \log -2$. The derivation system avoids this by the use of appropriate restrictions.

A similar problem arises in integration when trying to evaluate definite integrals such as $\int_{-1}^{1} \frac{1}{x^2-a^2}\,dx$. It is not enough to apply the symbolic rule, since the validity of the application of the rule depends on satisfying an antecedent. In this case the antecedent requires, among other things, that the expression being integrated is defined across the entire interval of integration.

While the integrands algebraic form adds the restriction $x \neq a$, we must also know that since this expression occurs in the context of an integral, $a \notin [-1, 1]$ must be true as well. Unless the derivation system is sensitive to issues of domain which determine whether or not a certain rule is applicable, the system will allow students to produce meaningless results that cannot but add to their confusion.

5.4 Logical knowledge required

In view of the role that restrictions play in the derivation system, the question arises as to how much logical knowledge the students require in order to be able to use the derivation system. Clearly at a minimum they must be aware of what a derivation is and that the lines are logically connected since students who use the derivation system are essentially constructing proofs.

From the students' point of view, however, these are informal proofs, fairly rigorous, but not numbingly explicit. It is not pedagogically justifiable to require students to be aware of all axioms and rules of inference for the predicate calculus in order to work algebra problems. Students should be made aware only of that which is mathematically necessary for their understanding of the content and effective use of the system. Consequently, though the system internally uses much logical machinery to track the interdependencies of the lines of work, students need only have a passing familiarity with the most natural and mathematically essential aspects of logical deductions in order to use the system.

5.5 The underlying engine?

At this point we would like to address a question which arises naturally out of the above critiques of commercial symbolic computation packages, namely whether or not they have any use at all within a system designed for students. As we stated in the beginning of this paper, we do not dispute the claim that such programs can play important roles in enabling students to explore mathematical concepts. What we wish to examine is whether commercial programs have a place in the role played by our derivation system.

There are two separate questions to be addressed here depending on how wide a role one wishes to assign to the program. If one is asking whether something like Maple is

suitable by itself for this role, the answer will have to be a resounding no. For reasons articulated by ourselves throughout this paper, (for similar arguments see the contribution of Beeson in this volume and Kajler and Soiffer 1996), Maple is clearly not up to the task unaided. The question therefore becomes whether a commercial system can have any place within a pedagogically adequate system.

This second question is more suble. In his paper in this volume, Beeson gives arguments that the answer to this question should also be no. While we appreciate his arguments we do not share in his conclusion. In fact our derivation system as it currently stands makes use of the C-callable kernal of Maple V release 2. This gives us access to the vast knowledge built into the Maple library. We have built our own semantic machinery but we have not developed a full blown symbolic computation system from scratch.

The reasons underlying our decision for this our two fold. The first is our recognition that programs like Maple represent massive programming efforts coupled with the feeling that to repeat such an effort would be a waste of resources. It is our belief that, suitably used, such symbolic computation programs provide a powerful resource.

Exactly how powerful a resource they provide is something that can only be determined after the derivation system has been completed. It may be the case that we will find that unless we have complete control off all computations from start to finish we cannot preserve all the information we need. In this case we will find ourselves at the end of the day having supplied all the features of Maple that we use and our position will have converged with Beeson's.

Our second reason for using Maple stems from the realization that when students complete this course and move into more advanced courses they will find themselves working with existing symbolic computation programs. To prepare them for this the course must ween them from their dependency on the error checking provided by the derivation system and help them develop their own ability to act as their own auditors. By incorporating Maple into our derivation system and giving them access to it through the course we hope to facilitate their eventual transition to regular users.

6 Two examples

The following two examples illustrate how the derivation system is used. The first example shows a problem from algebra and discusses how students at different levels might solve it. The second example shows a problem from the integral calculus.

6.1 Example 1

6.1.1 Problem

To factor $2x^3 + x^2 - 18x - 9$

6.2 A First Year Algebra Student

A first year algebra student encountering a problem like this has just learned about combining like terms, and grouping. Consequently, he or she should attack this problem

by grouping terms, to see if common factors exist. In this case, one could highlight $2x^3 + x^2$, then click on the "add parentheses" rule in the grouping rule group. One may then factor out x^2 simply by highlighting the terms in the parentheses, holding the SHIFT key down while highlighting x^2 (or typing x^2 in the INPUT window), and applying the "Factor out an expression" rule. This yields $x^2(2x + 1)$.

Similarly, one groups the $-18x - 9$ terms together, and factors out -9 to yield $x^2(2x + 1) - 9(2x + 1)$. Finally, using either the "Combine Like Terms" rule or the "Distributive" rule (both in the grouping rule group) with the entire expression highlighted will give $(x^2 - 9)(2x + 1)$. (One could also highlight the whole expression, then highlight just one of the $2x + 1$ terms with the SHIFT key held down, and apply the "Factor out an expression" rule.

At this point, one may factor $x^2 - 9$ in three ways:

1. by using the "Difference of squares" rule (after replacing 9 by 3^2) or
2. by typing one of its factors in the input window and using the "Factor out an expression" rule.
3. if one is far enough in the course to have received the quadratic formula, one may apply the "Factor quadratic" rule.

If one noticed a different pattern, one could solve the original problem with this same method after first transposing the middle two terms. Highlight the $-18x$ term and hit the left arrow key: the result will be $2x^3 - 18x + x^2 - 9$. One may factor $2x$ out of the first two terms in the way described above, then group the last two terms to get $2x(x^2 - 9) + (x^2 - 9)$ and proceed as before.

6.3 A Second Year Algebra Student

A somewhat more sophisticated student might solve the problem by directly dividing out factors. To do this, one highlights the entire expression, types a factor into the input window, and applies the "Factor out an expression" rule.

If one sees that 3 is a root of the polynomial, for instance, one could factor out $x - 3$. Selecting the polynomial, typing $x - 3$ into the input window, and applying the "Factor out an expression" rule yields $(x - 3)(2x^2 + 7x + 3)$. One could then factor $2x^2 + 7x + 3$ in a variety of ways:

1. If one sees a factor, one may apply the "factor expression" rule again
2. One may complete the square. First factor out 2 using the "factor expression" rule. This gives $2\left(x^2 + \frac{7}{2}x + \frac{3}{2}\right)$. Then replace $\frac{3}{2}$ by $\frac{3}{2} + \frac{49}{16} - \frac{49}{16}$, and use the arrow keys: $2\left(x^2 + \frac{7}{2} + \frac{49}{16} + \frac{3}{2} - \frac{49}{16}\right)$. Use the "Add parentheses" rule to put the perfect square together, then apply the "Perfect square" rule. We now have

$$2\left(\left(x + \frac{7}{4}\right)^2 + \frac{3}{2} - \frac{49}{16}\right)$$

Now highlight the $\frac{3}{2} - \frac{49}{16}$ and use the "Arithmetic" rule. This gives

$$2\left(\left(x+\frac{7}{4}\right)^2 - \frac{25}{16}\right)$$

which can be simplified further by applying the "Difference of squares" rule (after replacing $\frac{25}{16}$ by $\left(\frac{5}{4}\right)^2$. Now we have

$$2\left(x+\frac{7}{4}+\frac{5}{4}\right)\left(x+\frac{7}{4}-\frac{5}{4}\right)$$

which, after using the "Arithmetic" rule, becomes

$$2(x+3)\left(x+\frac{1}{2}\right) .$$

3. One may apply the "Factor quadratic" rule.

A student in a high-level course who needs to factor this polynomial, might use any of the above methods; but he or she would also have access to the "MAPLE factorize" rule, which can perform the factorization in one fell swoop, leaving him or her to focus on the application of the factorization to whatever the real problem at hand might be. Similarly, the "Factor quadratic" rule is usable only by students in second year algebra, as it applies the quadratic formula. Students at the beginning of the algebra course are not allowed to use even the "Factor out expression" rule until they demonstrate a proficiency with factoring using still more basic rules.

6.4 Example 2

6.4.1 Problem

To evaluate $\int e^{2x}\cos(3x)\,dx$.

6.4.2 Solution

We begin with the definition of a function:
1. $f(x) = e^{2x}\cos(3x)$

We click on the *Integral* rule group and choose the rule "Antidifferentiate both sides." This gives us the equation:

$$\int f(x)\,dx = \int e^{2x}\cos(3x)\,dx$$

which we save as equation 2. in the history window.

We now choose the "integration by parts" rule in the *Integral* rule group. We need to highlight the term that will be $\frac{dv}{dx}$ in the integration by parts formula $\int u\frac{dv}{dx}\,dx =$

$u \cdot v - \int v \frac{du}{dx}\, dx + C$. If we choose $\cos(3x)$ (by holding down the mouse button and passing over the term) we will get:

$$\int f(x)\, dx = e^{2x} \cdot \frac{1}{3}\sin(3x) - \int \frac{1}{3}\sin(3x) \cdot 2e^{2x}\, dx + C$$

We could choose to save this to our history window as is, but we instead do some additional simplification first. Using the arrow keys and "arithmetic" rule we collect the coefficients in the integral:

$$\int f(x)\, dx = e^{2x} \cdot \frac{1}{3}\sin(3x) - \int \frac{2}{3}\sin(3x) \cdot e^{2x}\, dx + C$$

And now, the "Extract a constant factor" *Integral* rule gives us:

$$\int f(x)\, dx = e^{2x} \cdot \frac{1}{3}\sin(3x) - \frac{2}{3}\int \sin(3x) \cdot e^{2x}\, dx + C$$

Again, we could click on the "save" button, but it isn't really necessary.

We now apply integration by parts again, and again choose the trigonometric function inside the integral (this time it's $\sin(3x)$) to play the role of $\frac{dv}{dx}$ (if we choose the exponential function, we will simply undo all our work!). Now we have:

$$\begin{aligned}\int f(x)\, dx &= e^{2x} \cdot \frac{1}{3}\sin(3x) - \frac{2}{3}\left(-e^{2x} \cdot \frac{1}{3}\cos(3x)\right. \\ &\quad \left. - \int \left(-\frac{1}{3}\cos(3x) \cdot 2e^{2x}\right)\, dx + C_1\right) + C\end{aligned}$$

This time, we'll save our formula as line 3. in the history window–in case we go wrong in the simplification, we can return to this without having to redo our work. Now we will do more simplification. First, as before, we bring the $-\frac{2}{3}$ out of the integral.

$$\begin{aligned}\int f(x)\, dx &= e^{2x} \cdot \frac{1}{3}\sin(3x) - \frac{2}{3}\left(-e^{2x} \cdot \frac{1}{3}\cos(3x)\right. \\ &\quad \left. + \frac{2}{3}\int \cos(3x) \cdot e^{2x}\, dx + C_1\right) + C\end{aligned}$$

The integral which remains is the same as the right hand side of line 2. in the history window: that is, it is equivalent to $\int f(x)\, dx$! To make the substitution, we must rearrange the terms inside our integral to match the order of the terms in the integral in line 2. This is done with the arrow key. Next, we highlight the whole integral, select line 2 and choose the "Substitute a value for an expression" rule from the *Substitution* rule group. This gives us:

$$\begin{aligned}\int f(x)\, dx &= e^{2x} \cdot \frac{1}{3}\sin(3x) - \frac{2}{3}\left(-e^{2x} \cdot \frac{1}{3}\cos(3x)\right. \\ &\quad \left. + \frac{2}{3}\int f(x)\, dx + C_1\right) + C\end{aligned}$$

We save this as line 4. and now need only do some basic algebra to finish. We use the distributive rule (in the *Grouping* rule group) to multiply out $-\frac{2}{3}\left(-e^{2x}\cdot 13\cos(3x)+\frac{2}{3}\int f(x)\,dx+C_1\right)$). We now have:

$$\begin{aligned}\int f(x)\,dx &= e^{2x}\cdot\frac{1}{3}\sin(3x)+\frac{2}{3}e^{2x}\cdot\frac{1}{3}\cos(3x)\\ &\quad -\frac{2}{3}\cdot\frac{2}{3}\int f(x)\,dx-\frac{2}{3}C_1+C\end{aligned}$$

Then, we highlight the $-\frac{2}{3}\cdot\frac{2}{3}\int f(x)\,dx$ and use the arrow keys to move it to the left hand side of the equals sign:

$$\begin{aligned}\int f(x)\,dx-\left(-\frac{2}{3}\cdot\frac{2}{3}\int f(x)\,dx\right) &= e^{2x}\cdot\frac{1}{3}\sin(3x)\\ &\quad +\frac{2}{3}e^{2x}\cdot\frac{1}{3}\cos(3x)-\frac{2}{3}C_1+C\end{aligned}$$

Again, we turn to the *Grouping* rule group. The "Cancel minus signs" rule will bring us to:

$$\int f(x)\,dx+\frac{2}{3}\cdot\frac{2}{3}\int f(x)\,dx=e^{2x}\cdot\frac{1}{3}\sin(3x)+\frac{2}{3}e^{2x}\cdot\frac{1}{3}\cos(3x)-\frac{2}{3}C_1+C$$

And the distributive rule, applied first to the entire left hand side, then to the first two terms of the right hand side, will give us:

$$\int f(x)\,dx\cdot(1+\frac{2}{3}\cdot\frac{2}{3})=e^{2x}\cdot\frac{1}{3}\left(\sin(3x)+\frac{2}{3}\cos(3x)\right)-\frac{2}{3}C_1+C$$

We may use the arithmetic rule to replace $(1+\frac{2}{3}\cdot\frac{2}{3})$ by $\frac{13}{9}$; the *Equation* rule group gives us a rule to divide both sides by 13/9; the *Fraction* rule group gives us rules to cancel and simplify terms in the fractions to yield at last

$$\int f(x)\,dx=\frac{3}{13}e^{2x}\cdot\left(\sin(3x)+\frac{2}{3}\cos(3x)\right)-\frac{6}{13}C_1+\frac{9}{13}C$$

We click on the "QED" button and are told:

> Well Done. You have correctly solved the problem.

There are several things to notice about the derivation above:

- The system ignores the constants of integration in determining the correctness of the answer. Doing the problem a different way, say by letting e^{2x} be $\frac{dv}{dx}$ for both applications of integration by parts, might well result in different constants of integration.
- There is a great deal of flexibility in which lines are saved in the history window. It was necessary to use one early line later in the proof for substitution, but the other lines were saved only for convenience (in case one wanted to start over from a certain stage in the problem) or clarity.

- Much algebraic simplification is optional. For this problem, any algebraic answer whose derivative is $e^{2x}\cos(3x)$ will be accepted. Most simplification, as in off-line work, is for the benefit of the student's vision. Of course, some simplification was necessary to justify the substitution of the original integral back into the equation.

7 What is gained

Having described the derivation system and explained how it addresses several deficiencies found in common symbolic computation packages, we turn to an examination of the pedagogical benefits derived from it. These benefits, coupled with the semantic processing of student input discussed above, are the true benefits afforded by symbolic computation.

7.1 Instant feedback with flexibility

The most immediate benefit of incorporating the derivation system into a mathematics course is that it allows students to get instant feedback on their work. This is true both with free response answers solved by students without use of the derivation system, and with answers produced in the derivation system. In traditional courses, several days may pass before a student receives graded work back from an instructor, which may allow erroneous patterns of reasoning to become settled. With immediate answer checking—which extends to checking each step for problems done entirely in the derivation system—students can recognize their misunderstandings as soon as they develop and seek help to correct them.

Instant feedback in computer-based courses does not absolutely require the use of symbolic computation software; however, such software allows semantic-based checking of answers and maximum flexibility of response. In a similar way, step-by-step evaluation does not absolutely require the use of a derivation system. Some courses try to evaluate students on a step-by-step level by specifying the steps to be applied and prompting students to type in each intermediate result. In essence, this decomposes complex problems into simple problems, each of which must be answered by the student. Yet there may be many correct approaches to solving a complex problem, and many reasonable choices for intermediate results. That students will take wildly different paths in reaching the goal in a derivation has been well documented in both our logic and set theory courses (Kane 1981; Suppes 1981). A derivation system allows step-by-step evaluation while allowing students to decide for themselves what steps to take in solving a problem.

7.2 Forced demonstration of understanding

In addition to focusing on what has been recently learned and in giving students instant feedback, the derivation system plays a crucial role in moving beyond the traditional limitations in evaluating student understanding. Traditional computer instruction evaluates a student's performance solely in terms of whether he or she produces correct an-

swers. It does not examine the process that the student went through to produce those answers. As such, it cannot adequately assess student understanding. A student, especially in mathematics, should always be able to justify how an answer was arrived at.

Forcing students to solve problems in the derivation system by applying rules is one way to evaluate whether or not the students understand the process of how to arrive at the correct answer. If the correct answer is given and the steps taken en route to the answer are all justified, then the student in some sense understands how to produce the answer. This type of evaluation of understanding is what exercises in derivation systems normally afford.

More than this is desirable, however, since within a derivation system arriving at answers does not entail that one completely understands the computational process required to produce the answer. This concern is particularly urgent in courses like algebra, in which one major objective (though by no means the only one) is to teach students how to perform a certain class of computation. To some degree the measure of ones knowledge of algebra is simply how well one can do algebraic computations. While this is certainly not the only criterion for understanding, it is often the sole criterion used on standardized examinations. It is essential to make sure that students not only know which rules might be appropriate in a particular situation, but also are able to apply the transformations themselves.

As mentioned above we achieve this within the derivation system by requiring students to supply partial result after they have applied rules. The system displays a partial form of the result and asks the students to supply the missing part. By forcing a student to do this frequently at first and periodically throughout the course, we make sure that the student can do the computation embodied in the rule.

8 Limitations and desiderata

The EPGY software is constantly under review, as feedback from our students pours in daily. To support our existing student base, we attempt to improve our courses and address observed deficiencies as quickly as possible. At the realities of managing hundreds of students of varying degrees of technical sophistication, each running our software on his or her own computer with its own particular idiosyncrasies make the potential consequences of implementing hastily improvised solutions quite unpleasant. As the number of students using our software has grown we have come to learn the importance of the careful collection and analysis of data concerning how students actually use the program. The importance of usability testing discussed by Avitzur in his contribution to this volume is a lesson we have learned well and often over the past few years.

There are many improvements we hope to add to the derivation system in the next few years. Most immediately, we intend to improve the range and quality of error messages, offering students who attempt to misapply rules a more complete explanation of the nature of their error, together with suggestions on what to do instead. Expanding our existing help system to contain pointers to appropriate places in the course would also help acheive this goal.

Along these lines we would also like to build student proofs directly into the derivation system. At the moment students who wish to see a complete proof of a derivation

currently must contact their instructors. One way, requiring little programming effort but a great deal of editing, would be to hard-wire solutions to each assigned problem in each course; another way would be to extend the system to generate automated solutions to wide classes of problems.

Still another method, intriguing in its use of student data, would be to collect and compare solutions already generated by students running the course and select the best (measured by some automated criteria) to be included in future releases of the course. Incentives could be offered to inspire students to come up with the shortest, or most elegant solution, but they key point is that as this data is already being collected and analyzed, it would be a natural to automatically incorporate such proofs into subsequent versions of the program.

There are improvements we would like to make involving speech recognition and handwriting input which we mention here without elaboration.

8.1 Validity v. vacuity

As mentioned above we ensure the literal validity of the derivation by including all contraints and dependencies needed for the valid application of each rule used as additional lines of the derivation. In this way, any finished derivation must be literally correct, as assumptions sufficient for each step are included as hypotheses in the history window.

In doing this one faces a trade off between halting a student derivation when one detects that the application of a rule leads to a contradiction and allowing the derivation to continue as true but vacuous. While the ideal is to detect errors when possible, we can not catch all such errors. It is possible to derive a seemingly false result; however, if one carefully checks the restrictions listed in such a derivation one will discover that the domain for which the result holds is empty. The result is logically valid, but may be misleading.

This difficulty is one common to any effort to guarantee the validity of student derivations. Indeed the work done by Beeson faces a similar problem, for as many clerical restrictions in MATHPERT are displayed only when requested, it may be possible to derive a seemingly false result *without* seeing the internally stored restrictions that invalidate it.

As we collect data on what sorts of contradictions arise most frequently we hope to improve our ability to detect them when they arise. Even if one cannot in principle notice all contradictions, one may be able to detect most of the ones which arise naturally in the specific body of curriculum for which our derivation system has been designed.

8.2 Forced justification of results

A more extreme way to test the ability of students to both compute and to justify their answers is to transform the derivation system into what we have dubbed a “justification system.” A justification system inverts the normal order of the derivation system in requiring students to first perform a computation themselves, and then to justify the computation by selecting the appropriate derivation system rule as justification. The derivation system is then used to evaluate the justification to make sure that the com-

putation performed by the student was correct. This style of work is reminiscent of the traditional two column style of giving geometry proofs where students are asked to list there statements in the left column and to give the reason justifying the statement in the right column.

While a system like this would be tedious to use if one had to input extensive mathematical expressions with a keyboard, it would not be so if one could input answers with a pen and tablet, using full handwriting recognition of two dimensional mathematical notation. With this method of input, working in the justification system would be essentially the same as working on paper the way students normally do when forced to show all of their work. By forcing the students to cite the rule which licenses their inference, the derivation system could verify that the student's computation was in fact correct. This would provide the students with the ideal environment in which to do assignments. They would be allowed to progress until they made an error at which point they would be asked to explain themselves. When correct they would be allowed to continue but when incorrect they would arrive at the source of their error. This has definite advantages over the traditional model for student homework and instructor response.

8.3 Automated solution generation

Another desideratum that we have yet to investigate is the automatic generation of solutions to derivations that are appropriate to student levels of understanding. Some promising work has been done by those working in the field of artificial intelligence in education, particular those working on intelligent tutoring systems (Nicaud 1994; Beeson 1990; Oliver and Zukerman 1991). The common approaches to these sorts of systems, combine symbolic computation environments with models of student problem solving to create tutorial systems in which students can work problems and receive instruction in the form of hints when they are stuck. They can also have the computer solve problems from some set of types.

In the introduction of Nicaud (1994), the author laments the fact that despite much work on intelligent tutoring systems since their inception in 1972, few are currently in use and few have been shown to have good teaching capability. He writes: "Many researchers who worked in this field have changed their goals and are now working on educative but non-teaching environments...Many researchers have more or less reached the conclusion that intelligent tutoring systems are *impossible.*" We at EPGY have not reached this negative conclusion. We have concluded, however, that prior to working on such systems one should accomplish two goals, namely:

- the development of a derivation system that allows students to solve problems in a natural way, without causing deviation from the strategies that they would apply when working with pen and paper; and

- the collection of a significant amount of data on how students actually reason about problems, particularly when required to show all their work.

As we deepen our understanding of student solutions we will begin to address this issue.

9 Final remarks

In this article we have discussed the role symbolic computation plays in an existing corpus of computer-based courses. While little in our work may be new from a theoretical standpoint, we have created what we believe to be the largest body of self-contained computer-based courses in mathematics at the secondary school level that are in general use. This in turn has enabled us to test the effectiveness of new techniques by seeing if they do in fact improve student results. As our courses are refined and we begin to analyze the data we have collected, we hope to deepen our understanding of what just exactly are the pedagogical ramifications of symbolic computation.

9.1 Availability

The software discussed herein is not commercially distributed. It solely for use by students taking courses through the Stanford University Continuing Studies Program from the Education Program for Gifted Youth. For up-to-date information about EPGY or the EPGY course software see `http://www-epgy.stanford.edu`.

References

Ager, T., Ravaglia, R., Dooley, S. (1989): Representation of inference in computer algebra systems with applications to intelligent tutoring. In: Kaltofen, E., Watt, S. (eds.): Computers and Mathematics. Springer-Verlag, New York, pp. 215–227.

Beeson, M. (1990): Mathpert: a computerized environment for learning algebra, trig, and calculus. Journal of Artificial Intelligence and Education 2: 1–11.

Brown, D., Porta, H., Uhl, J. (1991): Calculus and Mathematica: a laboratory course for learning by doing. In: Leinbach, L. C., Hindhausen, J. R., Ostebee, A. M., Senechal, L. J., Small, D. B. (eds.): The Laboratory Approach to Teaching Calculus. The Mathematical Association of America, Washington, DC. (MAA Notes, vol. 20).

Chuaqui, R., Suppes, P. (1990): An equational deductive system for the differential and integral calculus. In: Martin-Lof, P., Mints, G. (eds.): Proceedings of COLOG-88 International Conference on Computer Logic, Tallin, USSR, 1988. Springer-Verlag, Berlin Heidelberg, pp. 25–49.

Crandall, R. E. (1989): Mathematica for the sciences. Addison Wesley, Menlo Park, CA.

Cuasay, P. (1992): Cognitive factors in academic achievement. Higher Ed. Ext. Serv. Rev., 3(3). Higher Education Extension Service, New York.

Davenport, J., Siret, Y., Tournier, E. (1988): Computer algebra: systems and algorithms for algebraic computation. Academic Press, London, UK.

Hughes-Hallett, D., Gleason, A. M., et. al. (1992): Calculus. Wiley, New York.

ICTCM (1995): Eighth International Conference on Technology in Collegiate Mathematics. Prelimary Schedule. To be held Nov., 1995, Houston, TX.

Kaltofen, E., Watt, S. (eds.) (1989): Computers and mathematics. Springer-Verlag, New York.

Kane, M. T. (1981): The diversity in samples of student proofs as a function of problem characteristics: The 1970 Stanford CAI logic curriculum. In: Suppes, P. (ed.): University-level computer-assisted instruction at Stanford: 1968–1980. Inst. Math. Stu. Soc. Sci., Stanford University, Stanford, CA, pp. 251–276.

Kajler, N., Soiffer, N. (1996): A survey of user interfaces for computer algebra systems. To appear in the Journal of Symbolic Computation. Preprint available as Technical Report #5 from RIACA, Kruislaan 419, 1098 VA Amsterdam, The Netherlands.

Moloney, J. M. (1981): An investigation of college-student performance on the 1970 Stanford CAI curriculum. In: Suppes, P. (ed.): University-level computer-assisted instruction at Stanford: 1968–1980. Inst. Math. Stu. Soc. Sci., Stanford University, Stanford, CA, pp. 277-300.

Nicaud, J. F. (1992): Reference network: a genetic model for intelligent tutoring systems. In: Frasson, C., Gauthier, G., McCalla, G. I. (eds.): Proceedings of Intelligent Tutoring Systems (ITS'92), Montreal, Canada. Springer-Verlag, Berlin, pp. 351-359 (Lecture notes in computer science, vol. 608)

Nicaud, J. F. (1994): Building ITSs to be used: lessons learned from the Aplusix project. In: Mendelson, P., Lewis, R (eds.): Proceedings of Lessons from Learning, IFIP Workshop, Archamps, France (1993). IFIP Transactions A, 46: 181–198.

Oliver, J., Zukerman, I. (1991): DISSOLVE: an algebra expert for an intelligent tutoring system. In: Lewis, R., Otsuki, S. (eds.): Proceedings of Advanced Research on Computers in Education, IFIP TC3 International Conference, Tokyo, Japan (1990). North Holland, Amsterdam, pp. 219–224.

Ravaglia R. (1995): Design issues in a stand alone multimedia computer-based mathematics curriculum. In: Fourth Annual Multimedia in Education and Industry, Asheville, NC, pp. 49-52.

Ravaglia, R., de Barros, J. A., Suppes, P. (1994): Computer-based advanced placement physics for gifted students. Computers in Physics, 9:380–386.

Ravaglia, R., Suppes, P., Stillinger, C., Alper, T. (1995): Computer-based mathematics and physics for gifted students. Gifted Child Quarterly, 39: 7–13.

Richardson, D. (1968): Some undecidable problems involving elementary functions of a real variable. Journal of Symbolic Logic, 33: 515–521.

Suppes, P. (ed.) (1981): University-level computer-assisted instruction at Stanford: 1968–1980. Inst. Math. Stu. Soc. Sci., Stanford University, Stanford, CA.

Suppes, P., Sheehan, J. (1981): CAI course in axiomatic set theory. In: Suppes, P. (ed.): University-level computer-assisted instruction at Stanford: 1968–1980. Inst. Math. Stu. Soc. Sci., Stanford University, Stanford, CA, pp. 3–80.

Suppes, P., Sheehan, J. (1981): CAI course in logic. In: Suppes, P. (ed.): University-level computer-assisted instruction at Stanford: 1968–1980. Inst. Math. Stu. Soc. Sci., Stanford University, Stanford, CA, pp. 193–226.

Wagon, S. (1991): Mathematica in action. Freeman, San Francisco.