J. Symbolic Computation (1995) **11**, 1–000

A Survey of User Interfaces for Computer Algebra Systems

NORBERT KAJLER† AND NEIL SOIFFER‡§

†*RIACA, Kruislaan 419, 1098 VA Amsterdam, The Netherlands and Projet SAFIR, CNRS - INRIA - I3S, Université de Nice-Sophia Antipolis, Bat. 4, 250 rue Albert Einstein, Sophia-Antipolis, F-06560 Valbonne, France. Previously: ICM, Dept. of Mathematics and Computer Science, Kent State Univ., Kent, OH 44242, U.S.A.*

‡*Wolfram Research, Inc, 100 Trade Center Drive, Champaign, IL 61820, U.S.A.*

(Received June 22nd 1994)

Accepted for publication in the Journal of Symbolic Computation

This paper surveys work within the Computer Algebra community (and elsewhere) directed towards improving user interfaces for scientific computation during the period 1963–1993. This paper is intended to be useful to two groups of people: those who wish to know what work has been done and those who would like to do work in the field. The paper contains an extensive bibliography to assist readers in exploring the field in more depth. Work related to improving human interaction with computer algebra systems is the main focus of the paper. However, the paper includes additional materials on some closely related issues such as structured document editing, graphics, and communication protocols.

1. Introduction

There are several problems with current computer algebra systems (CASs) that are interface-related. These problems include: the use of an unnatural linear notation to enter and edit expressions, the inherent difficulty of selecting and modifying subexpressions with commands, and the display of large expressions that run off the screen. These problems may intimidate novice users and frustrate experienced users. The more natural and intuitive the interface (the closer it corresponds to pencil and paper manipulations), the more likely it is that people will want to take advantage of the CAS for its ability to do tedious computations and to verify derivations. However, unlike pencil and paper, CAS interfaces can be interactive so that many new and interesting ways of solving problems are possible.

§ E-mails: kajler@can.nl; soiffer@wri.com.

0747–7171/90/000000 + 00 $03.00/0

The scope of this paper is to present a survey of the efforts undertaken during the last 30 years within the Computer Algebra community (and beyond) to improve user interfaces in this area. Our goal is to be useful to people who want to become more familiar with this topic, to know about results already achieved and future research directions, and to take advantage of the extensive bibliography we provide on the subject. We limit our presentation to works directly related to human interaction with computer algebra systems. However, we have included additional material on closely related issues such as structured document editing, graphics, educational software, and communication protocols, with no intention to be exhaustive in these areas.

The core of this paper is a historical survey which covers the period 1963–1993, followed by additional materials highlighting various aspects of the problem. The historical survey itself is divided into four sections: Computer Algebra, Numerics, Document Processing, and Artificial Intelligence & Education.

2. Previous Surveys and Overview Works

The problem of improving user interfaces for CAS was highlighted a long time ago. In 1963, Minsky suggested in his Mathscope proposal (Minsky, 1963) some general directions to manipulate mathematical expressions on a computer screen. In Sammet (1969), section IV.7 presents "languages for numerical scientific problems with fairly natural mathematical notation" while section VII.7 is devoted to "formal algebraic manipulation languages requiring special equipments". These two subsections present the earliest prototypes allowing input and display of typeset mathematical expressions on batch processing systems.

Since that date, many discussions and publications have helped to define the needs of a "good" CAS user interface (Wells & Morris, 1972; Foster, 1984b; Arnon, 1987; Katz, 1987; Kajler, 1990; Kajler, 1995; Kajler & Monagan, 1996). Recently, two PhD theses were defended in that area. In Soiffer (1991), Soiffer exposed the results of his work on algorithms for efficient parsing, selection and display of mathematical expressions and the implementation of the MathScribe user interface. In Kajler (1993a), Kajler exposed the results of his work concerning the design of extensible and distributed CAS environments and the implementation of the CAS/PI user interface. Both theses include an overview of previous work in these areas and an extensive bibliography. Lastly, a survey on commercially available CASs (Foster, 1993) highlighted improvements that have been made to their user interfaces and remaining areas of weakness.

3. History

The first CASs were developed on batch processing systems before the advent of time-sharing systems. Input was supplied on punched cards, and the output was printed (usually some time later) on a line printer. The output could contain no special characters such as Greek letters or other mathematical symbols. Today, CASs are typically used on a time-shared computer system via an interactive terminal or on a workstation, but the form of the input and output has hardly changed during this time. Input is still a linear string of symbols (but it is typed on a keyboard now instead of a keypunch). In 1994 it is still the case that several of the major CAS's still form output using the same limitted character set, but it appears on a terminal screen instead of a printer.

Meanwhile, very sophisticated display of mathematical expressions has become possible using typesetting systems. These systems were developed mainly to allow the inclusion of mathematical expressions in papers and books, and little of this technology has been applied to interactive user interfaces. Until recently, there has been surprisingly little work done on graphical CAS interfaces.

Today, most computers and terminals include a graphical screen and a pointing device (mouse). These elements allow a more convivial use of the computer, based on the WIMP paradigm†.

To date, systems that allow two-dimensional input of expressions use one or more of the following mechanisms: *templates*, *overlays*, and *parsing*. Briefly, entering an expression via templates involves choosing a template representing some mathematical notation and then filling in the subparts of that template. For example, a user might choose a fraction template and then fill in the numerator and denominator of that template. Templates lead to a prefix style of entering expressions. Overlays are a variation on templates whereby a selected subexpression is substituted for one of the subparts of the template and then the entire template is substituted back in place of the original selected subexpression. Overlays allow a more natural infix style of input for many single-operator notations. Parsing involves using precedence relations to bind operands to operators and requires that every mathematical notation have a linearized form. In addition to allowing a natural infix style of input for single-operator notations, parsing allows a natural left-to-right ordering of input for multi-operator notations such as parentheses, integrals, and programming language constructs.

The rest of this section reviews previous and current user interface work related to scientific software. It is divided into work directed towards CASs, work directed towards numerical computation, work directed towards typesetting, and work directed towards problem solving and education. There has also been a great deal of work done in the area of programming environments that is relevant; (Soiffer, 1991) discusses that work in more detail.

3.1. COMPUTER ALGEBRA SYSTEM INTERFACES

The earliest system to provide two-dimensional notation for display was Clapp and Kain's Magic Paper I described in (Clapp & Kain, 1963). Magic Paper I was developed in 1963 at Bolt Beranek and Newman and ran on a PDP-1. It used a typewriter to input math expressions, a display scope to visualize typeset outputs and a light pen for input/output‡.

Minsky's Mathscope proposal (Minsky, 1963) is also dated 1963. Martin refined Minsky's ideas and they were eventually incorporated into the Symbolic Mathematical Laboratory (Martin, 1967), a precursor to MACSYMA (MATHLAB Group, 1977) at MIT. Input-output devices (teletype, plotter, scope, and light pen) were connected to a PDP-6 while symbolic manipulations were performed on a 7094, via a transmission device. On the scope, it was capable of displaying normal mathematical notation using different fonts and special characters such as an integral sign. It also allowed the use of a light pen

† WIMP stands for Window, Icon, Menu, and Pointer. This model was initially developed at Xerox PARC (Smith, Irby, Kimball, Verplank, & Harslem, 1982) during the seventies and was then popularized with the Macintosh (Apple Computer, Inc., 1991) computer in the middle of the eighties.

‡ Later, Clapp developed a new and independent version that retains the same name (Clapp, 1986). This second version ran on a modified PDP-1 with special input-output devices: a special keyboard, a flicker-free display, a light pen, a push-button panel, and two foot switches.

(a) Maple

```
                  sqrt(3) t
 2 pi  int(cos(---------), t)            t
                     2                  e
----------------------------- + -----------
                   (t/2)         f(t, 2/3)
              3 e
```

(b) MACSYMA

```
        /
        [     sqrt(3) t
2 %pi   I cos(---------) dt
        ]         2                t
        /                         e
------------------------- + -------
          t/2                     2
       3 e                  f(t, -)
                                  3
```

(c) TEX

$$\frac{2\pi \int \cos(\frac{\sqrt{3}\,t}{2})\,dt}{3e^{t/2}} + \frac{e^{t}}{f(t,2/3)}$$

Figure 1. Sample Output

to select output on the scope. The light pen was used to select variables and operators. Selecting an operator selected the smallest subexpression containing that operator.

Martin's work never made it out of the laboratory. This is probably due to the special hardware combination required to use the prototytpe and to the fact that at that time graphic display hardware was very expensive and not widely available. There was not much standardization of the available display devices, which meant that porting a graphics system to different hardware was very difficult. Later systems regressed from Martin's ground breaking system.

By 1978, many computer terminals were capable of random cursor positioning. Despite this, input for all CASs was restricted to a linear string of symbols. Most CASs improve upon one-dimensional output by displaying subscripts and superscripts (exponents) on different lines, and by displaying quotients vertically. An example of Maple (Char, Geddes, Gonnet, Leong, Monagan, & Watt, 1991) output is shown in Figure 1a†. Another improvement is to use "line printer graphics" to crudely represent some special symbols, such as integration signs. Figure 1b shows the expression from Figure 1a as it would be output by MACSYMA (MATHLAB Group, 1977). MACSYMA embodies more or less the state of the art in expression display for CAS on character output devices.

Two attempts at editors that understood MACSYMA's expression structure were implemented in 1979 and presented at the MACSYMA Users' Conference in Washington, D.C. Neither editor received much use and were abandoned until recently. They are described below.

MAC-ED (Fateman, 1979), written by Fateman at Berkeley, was the less ambitious of the two implementations and was designed to run on character terminals which were prevalent among the MACSYMA user community at the time. MAC-ED's interface was basically that of a line-oriented editor: selections, changes, etc., were displayed on a new

† Maple V, Release 2 takes advantage of modern window systems and is described later in this section. The same is true of some versions of MACSYMA and Mathematica.

line. Users typed commands to move around the expression and change it by giving part numbers. For example, the command `2` would select the second subexpression of the previous expression and the command `2:x+y` would replace the second subexpression of the previous expression by $x + y$. Commands were also provided that allowed a user to select a subexpression by searching for it. For example, the command `find(x+y, t)` selects the first occurrence of $x + y$ in the previous expression, regardless of the depth of the subexpression $x + y$. MAC-ED provided two-dimensional display of expressions with elision of detail.

The other editor, written by Hoffman and Zippel at MIT, was an Emacs (Stallman, 1979)-like editor for MACSYMA that was designed to run on cursor-addressable character terminals (Hoffman & Zippel, 1979a; Hoffman & Zippel, 1979b). The editor split the window into two parts: a window where expressions were displayed in their two-dimensional form (it used MACSYMA's output routines) and a buffer window where users entered commands and expressions. The editor included commands for defining, copying, deleting, and replacing a selection. It also had commands for walking the expression tree and for applying a MACSYMA command to the selection. Selections were indicated by drawing a box with asterisks around the selection (the expression was redrawn in-place—there was no intelligent update of the screen). As with Emacs, users could bind whatever function they wished to any key. This allowed frequently used and user-defined functions to be easily applied to expressions. In 1988, Symbolics reimplemented Hoffman and Zippel's editor to include mouse support and integrated it with the window system for the Symbolics 3600 series machines (Krausz, 1988). Taking advantage of the advanced graphic facilities provided by the Lisp Machine at that time, this MACSYMA interface offered user-reconfigurable menus and allowed selection of screen parts via the mouse.

In 1978, Foderaro (Foderaro, 1978) added to MACSYMA the capability of generating eqn (Kernighan & Cherry, 1978) commands and storing them in a file. The contents of this file can be incorporated into a paper and then run through eqn and troff. Fateman (Fateman, 1987) wrote a version for TEX without line breaking in 1987. Antweiler, Strotmann, and Winkelmann added a similar feature to Reduce in 1989 (Antweiler, Strotmann, & Winkelmann, 1989) that addresses line breaking.

Figure 1c shows the previous example as displayed by TEX and demonstrates how much more comprehensible typeset output is. To take advantage of this, Foster (Foster, 1984a) adapted Foderaro's program in 1983 so that the output was sent to eqn/troff immediately and then displayed on a bitmapped screen. The result was disappointingly slow—one to five minutes per page. Unaware of Foster's work, Leler (Leler & Soiffer, 1985) also wrote a similar program which had much better performance. Work on integrating typesetting systems and CASs was abandoned in favor of developing display programs from scratch for numerous technical reasons, some of which are mentioned in Section 4.4. Briefly, these include eqn's lack of a line breaking algorithm and the lack of access to troff's internal data structures which prevents the interface from knowing the position of expressions on the screen.

DREAMS (Foster, 1984a), written by Foster in 1984 as part of his master's project at Berkeley, could display a (single) MACSYMA output expression in a special window on a workstation screen using troff fonts. Anderson borrowed from Foster's work and implemented a system named EXED (Anderson, 1983) similar to DREAMS that also allowed selection of a subexpression with the mouse but was not connected to a CAS.

In 1985, Soiffer (Leler & Soiffer, 1985) produced the first interface after Martin that could handle both input and output. This system (simply named the "Reduce pretty printer") was connected to Reduce (Hearn, 1984) and split the (non-windowed) Tektronix workstation screen into two regions: a display region and a dialogue region. The relative sizes of these regions could be changed by users. Each expression was displayed in its own *strip*. The size of the strip was based upon the size of the expression. As each new expression was displayed, the old expressions were automatically scrolled up in the display region. The contents of any strip could be scrolled horizontally independently of the other strips using a joy-disk. The entire display region could also be scrolled up or down using a joy-disk.

Subexpressions of an expression could be selected by users and entered back into Reduce. A selection was made by pushing a mouse button and moving the cursor over a subexpression with the mouse. Similar to EXED, the smallest subexpression whose bounding box contains the cursor was highlighted. When the mouse button was released the highlighted subexpression was entered back into the input stream.

The Reduce pretty printer did not break large expressions into several lines. Instead, it used the interactivity provided by the workstation to allow users to scroll the large expression horizontally. Additionally, large expressions and subexpressions could be collapsed manually (selected via the mouse) or automatically by setting width and depth limits. The subexpressions could be expanded back to their full form at any time.

The limitations of a single display area and the unnaturalness of having a linear input form and a two-dimensional output form were the motivations for implementing MathScribe. An early version of MathScribe (Smith & Soiffer, 1986; Tektronix, Inc., 1988) was demonstrated at SYMSAC '86 in Waterloo, Canada in the summer of 1986. Briefly, MathScribe provides both two-dimensional input and output of expressions in a multi-window environment. Users can freely edit any displayed expression in-place and the expression is reformatted and displayed correctly after each keystroke. The primary mode of input is parser-based, but both templates and overlays are also supported through a collection of predefined menus. MathScribe also contains several features to aid in understanding large expressions. This includes horizontal and vertical scrollbars, local and global abbreviations, and elision. However, no attempt was included to perform automatic line-breaking. Figure 2 presents a sample MathScribe session.

Another system presented at SYMSAC '86 was PowerMath (Davenport & Roth, 1986) by Davenport and Roth. PowerMath ran on an Apple Macintosh and was somewhat limited in what it could do because of the necessity of running in less than 512K bytes of memory. Both input and output are one-dimensional. PowerMath's novel features include using specialized windows for input, output, function definitions, and for value definitions. The function and value windows define the environment in which the computation is performed. By opening and closing windows, differents results could be obtained.

Within a year after the SYMSAC '86 conference, several other systems were announced. The exact chronology is unclear. These systems included GI/S by Young (Young & Wang, 1987), Milo by Avitzur (Avitzur, 1988), CaminoReal by Arnon (Arnon, Beach, McIsaac, & Waldspurger, 1988; Arnon, Waldspurger, & McIsaac, 1987).

Young, as part of his Master's project at Kent State, produced an interface that was similar to the Reduce pretty printer in concept, but was more sophisticated in many ways (Young & Wang, 1987). GI/S was connected to MACSYMA and ran on a Tektronix workstation. GI/S had a line-oriented editor and history mechanism in the dialogue area and allowed multiple windows (instead of a single window) in the display area. Young's interface also included the ability to draw graphics in a window. A significant difference

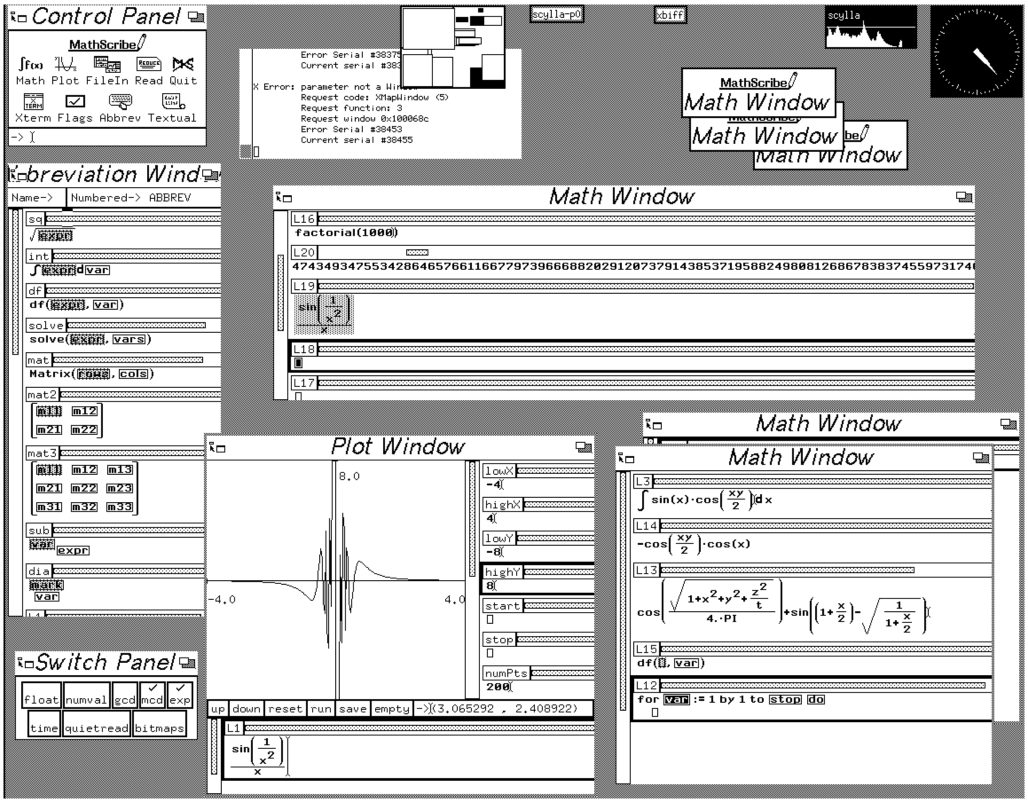

Figure 2. A sample MathScribe session

between GI/S and the earlier systems was that it used a display structure that was not necessarily related to the underlying algebra system's representation. This allowed a user to select any rectangular submatrix in an array or any consecutive characters in a linear expression, such as selecting $b + c*$ from the expression $a - b + c * d$. Another feature of GI/S is that alternate streams of computation could be handled in "scratch windows". All variables in a scratch window were local to that window. This was done by prepending the window name to all variables before sending the variables to MACSYMA, and stripping the prefix before printing the results.

Milo (Avitzur, 1988) was developed as an aid to solving undergraduate physics homework at Stanford where Avitzur was a student. It was developed for the Apple Macintosh and allows text, expressions, and simple plots to be included in a document. The text, expressions, and plots are on separate "lines" (i.e., vertically separated regions of the window). Milo is a rudimentary CAS and includes some basic simplification commands and a simple pattern matcher. Users can enter rules and have them selectively applied to an expression (or subexpression) in order to simplify or solve a problem. Milo allows two-dimensional input using the overlay model of input. Less powerful, but easier to use than most others CAS, Milo, as well as Theorist (presented later) and Kaava (Pasanen, 1992; Rimey, 1992) can be considered as mathematical assistants, a class of software more suitable to solve elementary mathematical exercises than addressing complex scientific problems. Several years later, parts of Milo were embedded in FrameMaker (Frame

Technology, Corp., 1989), a document processing system. This added not only expression editing to FrameMaker, but also symbolic manipulation of mathematical expressions. In 1994, parts of Milo were embedded in the Graphing Calculator (see § 4.7).

Milo used a slightly different method of selection than the systems described earlier. Instead of using the smallest box surrounding the mouse, the mouse is used to define a rectangle and the selection is the smallest box enclosing the rectangle. The button down position defines the upper-left corner of the rectangle and the position of the cursor with the button held down defines the lower right corner. This approach allows easier selection of contiguous subexpressions. However, because of its input model, Milo does not allow selection of isolated operators (for replacement or deletion). Milo maintains correct syntax at all times and defines a large number of operators, but is not extensible. It does not do line breaking or provide tools for dealing with large expressions (which are hard to generate without a CAS anyway). Milo was the first system to implement

$$\frac{\boxed{x}\times\mathrm{y}}{2}+x=1 \qquad \frac{\mathrm{y}\times\boxed{x}}{2}+x=1 \qquad \frac{\mathrm{y}}{2}\times\boxed{x}+x=1 \qquad (\frac{\mathrm{y}}{2}+1)\times\boxed{x}=1 \qquad \frac{\mathrm{y}}{2}+1=\frac{1}{\boxed{x}}$$

Figure 3. Example of the successive application of the MoveRight command of Milo

direct manipulation of expressions. Figure 3 shows an example of this: first x is selected, then while holding the option key down, the mouse is used to drag the expression to the right. The figure shows the result of dragging x further and further to the right.

CaminoReal was developed at Xerox PARC to explore the idea of "active documents". CaminoReal can be viewed as an extension to the Cedar environment's multimedia document editor Tioga (Swinehart, Zellweger, Beach, & Hagmann, 1986). CaminoReal adds not only the capability to enter and edit mathematical expressions to Tioga, but also the ability to perform computations on those expressions. With this facility, a user can mail a Tioga document to another user who can verify, evaluate, or change any expression or derivation in the document by invoking CaminoReal.

Entering and editing expressions is performed in a window separate from the main document using templates and/or overlays. Expressions can be freely copied to and from Tioga and are treated as atomic "glyphs" by Tioga and can appear anywhere any other character can appear. A flag is used to determine whether an expression should be displayed by Tioga "as is" or should be evaluated by CaminoReal before displaying.

Besides its integration with a document editor, CaminoReal introduces simultaneous access to several "algebra servers" including Reduce, SMP, and SAC-2. Users can request that a computation be performed by a CAS that resides on a remote machine. The CAS is started up anew for each computation requested. The CASs were not modified in any way: CaminoReal knows the names of the functions corresponding to each operator for each system. The connection is too primitive for serious use though. Its problems include: lack of saved state, unhandled warning and error messages, the inability to interrupt the computation, and the lack of debugging support for the remote system. Also, CaminoReal does not address concurrent use of the different servers or simultaneous use of the editor and algebra servers. In addition to the remote server capability, CaminoReal contains a small, domain-oriented CAS.

Mathematica (Wolfram, 1988) was designed to be used with different front ends. The first versions came with a standard terminal front end and a notebook front end. Early versions of Mathematica used MathTalk (Wolfram Research, Inc., 1988) as its communication protocol; later versions used a more general communication protocol called MathLink (Wolfram Research, Inc., 1993a) (discussed later). The notebook front end (Wolfram Research, Inc., 1993b) mixes text, equations, and graphics in a window (on separate "lines"), and is similar to Milo in this respect. However, the equations were neither typeset, nor editable. Notebooks distinguish between input, output, and text, and allow users to make a change and automatically re-evaluate all input so that they can see the effect of the change easily. Mathematica notebooks also have outlining: consecutive text, equations, etc., can be linked together and collapsed. These in turn can be linked and collapsed again. This allows users to easily skip irrelevant or uninteresting parts of the notebook. Notebooks can be used as active documentation for new functions (the expressions can be evaluated) or as active tutorials on various subject matter. An example of a Mathematica notebook is shown in Figure 4. Also, notebooks can be translated into PostScript and printed, allowing creation of scientific documents from Mathematica. Notebooks allow expressions to be associated with different Mathematica sessions if desired.

Many companies offer extensions to Mathematica. Among these are: Gourmet, a fancy calculator front end to Mathematica on NeXT computers, and HelpStack/ToolBook, a hypertext help package for Windows and Macintosh which eases retrieval of function definitions and examples (Hart, 1994).

MuMath (Rich & Stoutmeyer, 1979) was the first CAS available on a personal computer. Derive (Soft Warehouse, Inc., 1991) is its successor and is a very compact and easy to use system for the IBM PC (and compatibles). Derive is also the first CAS to run on a portable calculator (HP 95). While being limited by the basic facilities provided by standard personal computers (80x24 characters screen, limited memory, lack of pointing device, etc.), Derive allows selection of subexpressions by moving a cursor driven from the keyboard. In addition, a double menubar is available under the editing area that offers interactive access to every feature of the system: computation, display, graphics, etc.

One of the newest and most novel interfaces is Theorist (Bonadio, 1989). Theorist, which runs on a Macintosh, also contains a small CAS. Like Milo, Theorist supports text, expressions, and graphics. Theorist's graphics package has a sophisticated user interface that allows direct manipulation of the plots. Also, the plots are connected to an equation so that if you change the equation, the plot is redrawn as you make the change. Also, editing windows include an outliner with buttons to shrink/expand paragraphs or manipulate graphics.

In Theorist, expressions can be entered from the keyboard using either the line model or as in a programming language. Theorist also supports palette entry: several palette windows provide collections of predefined symbols and expression skeletons which can be entered via the mouse. The method used to select expressions is similar to that used by Milo. Theorist also allows multiple selections. In general, operations work on each selection independently.

Each window starts with a separate list of predeclared variables and functions (e.g., i and sin) which can be changed to support differing notations. Every variable and function used must be given a "type". Like PowerMath and GI/S, calculations in different windows can produce different results because of differing declarations.

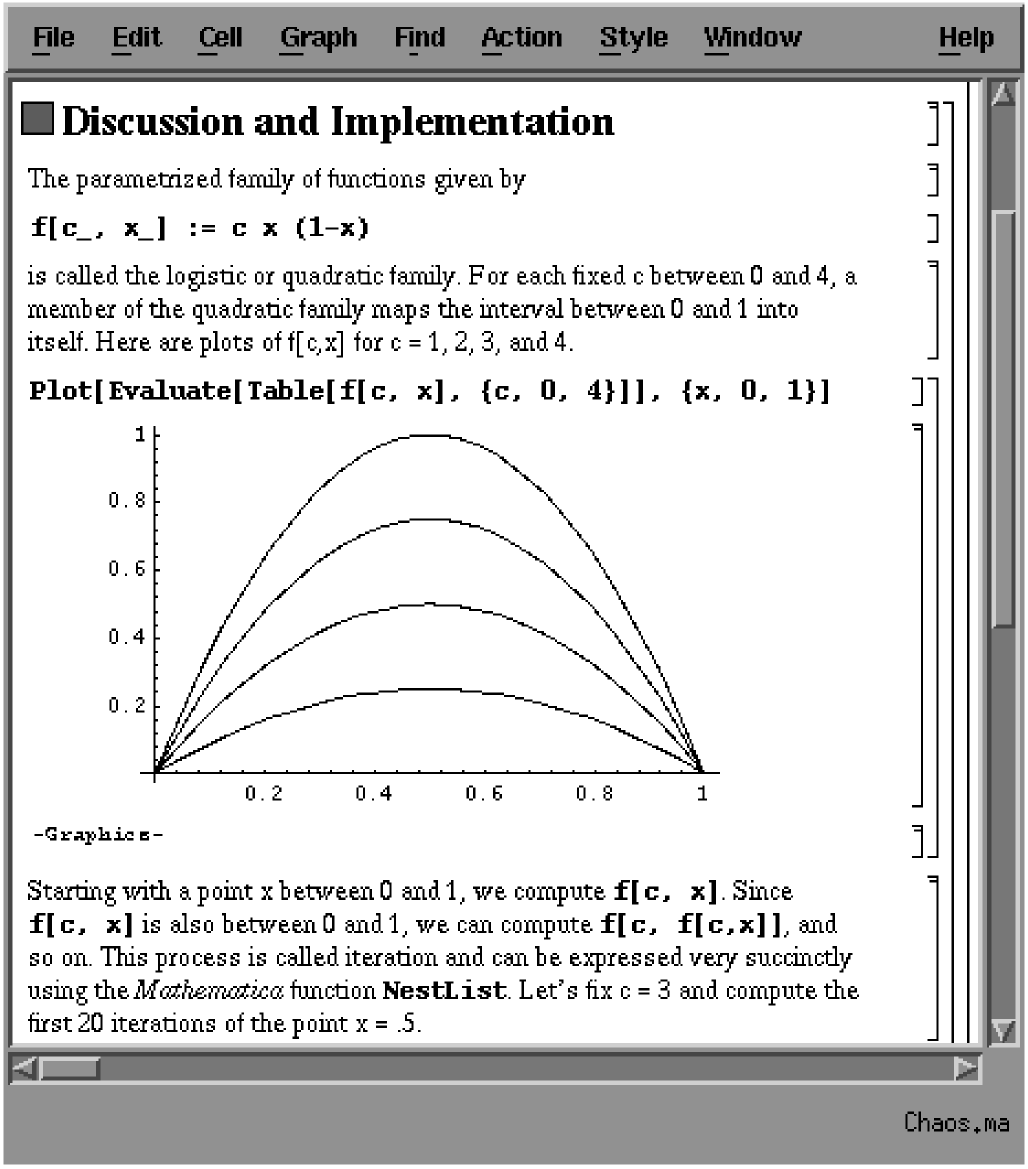

Figure 4. A sample mathematica notebook

Theorist allows users to commute subexpressions by direct manipulation: a subexpression is selected and can be "dropped" back into the expression at legal places. Similarly, subexpressions (including variables) can be selected and moved to one side of an equation with appropriate results. For example, if x is selected in the equation $ax + b = 0$ and moved to the right hand side, the equation becomes $\frac{-b}{a} = x$. Theorist attempts to

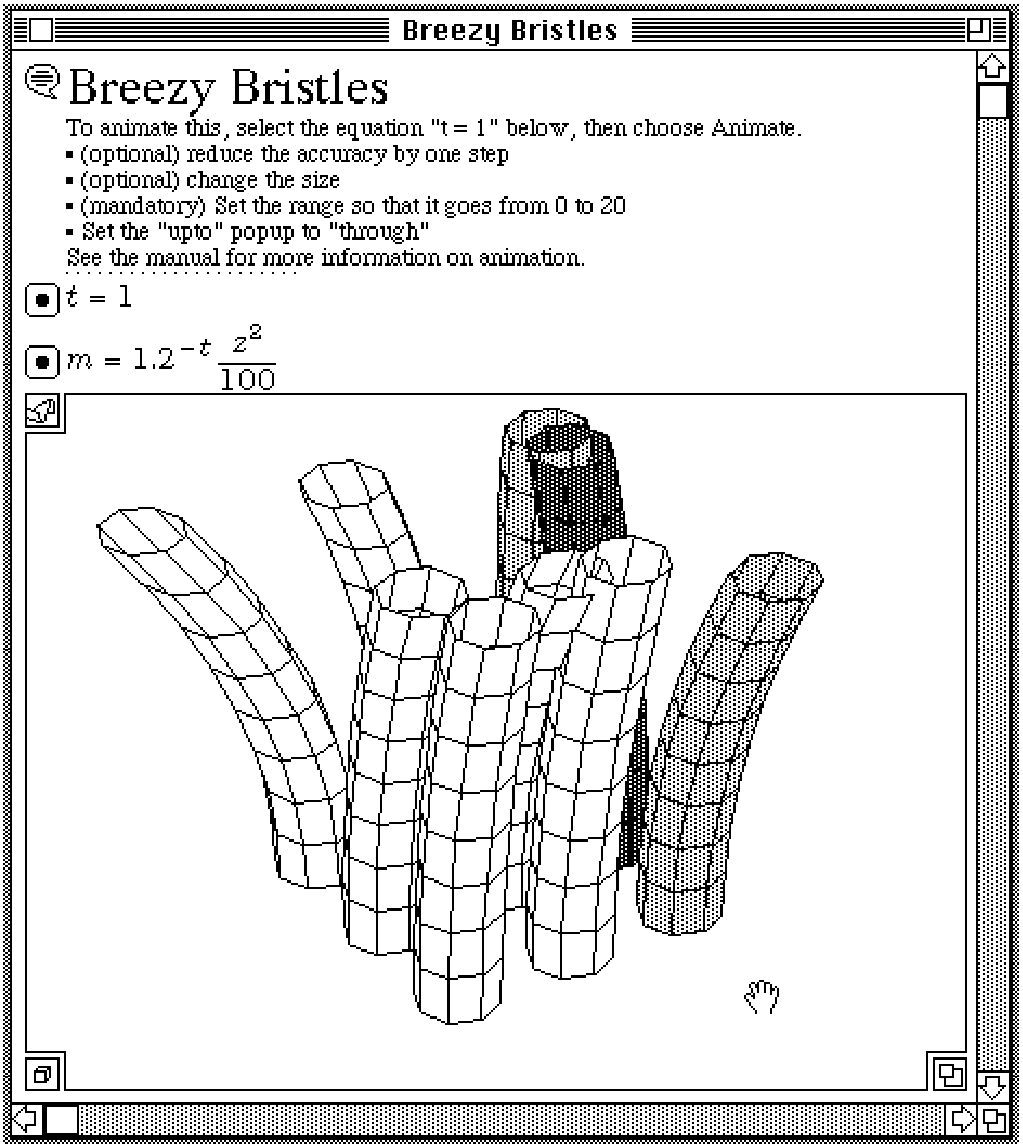

Figure 5. A sample Theorist session

perform only legal manipulations† based on the types of the variables being manipulated (e.g. two variables in a product that are declared to be matrices do not commute). Substitution is another direct manipulation that is supported. Substitutions are performed by selecting a defining equation and "dropping" it onto an occurrence of the defined variable.

† Determining the legality of a manipulation is quite difficult in general and is not always achieved in the current version of Theorist.

Theorist recognizes that large expressions can occur and allows users to collapse large expressions into ellipsis ("..."). Sums and products are treated specially and collapse into a single term/factor with ellipses on either side; selection of the ellipsis moves to the next term/factor. Auto elision is also supported. Figure 5 shows a sample Theorist session.

The successor of GI/S, SUI (Scientific User Interface) (Doleh & Wang, 1990) was initially presented in 1990 by Doleh as part of his PhD Thesis at Kent State University. As with GI/S, SUI uses different windows to separate input and output. Expressions are entered as linear text, while output is displayed graphically. SUI also allows different back ends to be used simultaneously. These include MACSYMA and REDUCE, a TEX code generator, and a surface plotting engine. With respect to CaminoReal, SUI adds concurrent use of the different servers and provides buttons to interrupt computations from the user interface. However, SUI does not perform conversions between the different formats used by available servers. For instance, expressions to be sent to MACSYMA and REDUCE have to be entered in separate windows using the system-specific command language. Also, expression templates available from menus in each input window may differ to fit the system-specific command language.

Commercially available from NAG since 1991, Axiom (Jenks & Sutor, 1992) is the direct successor of Scratchpad II system (Jenks, 1984) developed by IBM during the eighties. Axiom's user interface uses conventional linear input and character-based output. However, Axiom is the first CAS interface to include a *hypertext* facility. This facilitates browsing through the on-line documentation.

In 1992, new versions of Maple and MACSYMA were released that included significant improvements to their front ends. The Maple VR2 front end is similar to Mathematica's notebooks in that text and expressions can appear on separate "lines." Unlike, Mathematica notebooks, Maple's graphics appear in separate windows and there are no mechanisms to collapse and link paragraphs. An improvement over Mathematica's notebooks is that the output is displayed in typeset form by default, with the traditional output still available as an option. This work is based on Tyhurst Master's thesis (Tyhurst, 1993) which focuses on improved output presentation of mathematical expressions, including formula typesetting and line-breaking.

The new version of MACSYMA (Krausz, 1989), which runs under Microsoft's Windows environment, also provides typeset display of output. However, MACSYMA does not break expressions across lines; instead, expressions can be scrolled horizontally. In both Maple and MACSYMA, the expressions are output only and cannot be edited or selected. MACSYMA's new version also includes improved plotting and access to the manual via a brower.

Also in 1992, Vielhaber developed as part of his Master's thesis an experimental user interface for SACLIB built on top of the Interviews toolkit (Vielhaber, 1992). The user interface is a separate component that communicates with a computer algebra shell. The user interface requires expressions to be entered in prefix manner via the keyboard and mouse. Potentially, different shells can be started to communicate with several systems from a single editor.

Presented during ISSAC'92 in Berkeley, CAS/PI (Kajler, 1992b; Kajler, 1993b) is a CAS graphic user interface designed to be highly portable and extensible. CAS/PI was developed by Kajler as part as his PhD thesis.

Most components of CAS/PI are derived from high level specifications expressed in some ad-hoc languages. The components include a generic formula editor and a software bus architecture allowing local and remote software components to communicate by message broadcasting. An important goal of CAS/PI is to allow *expert users* to tailor the

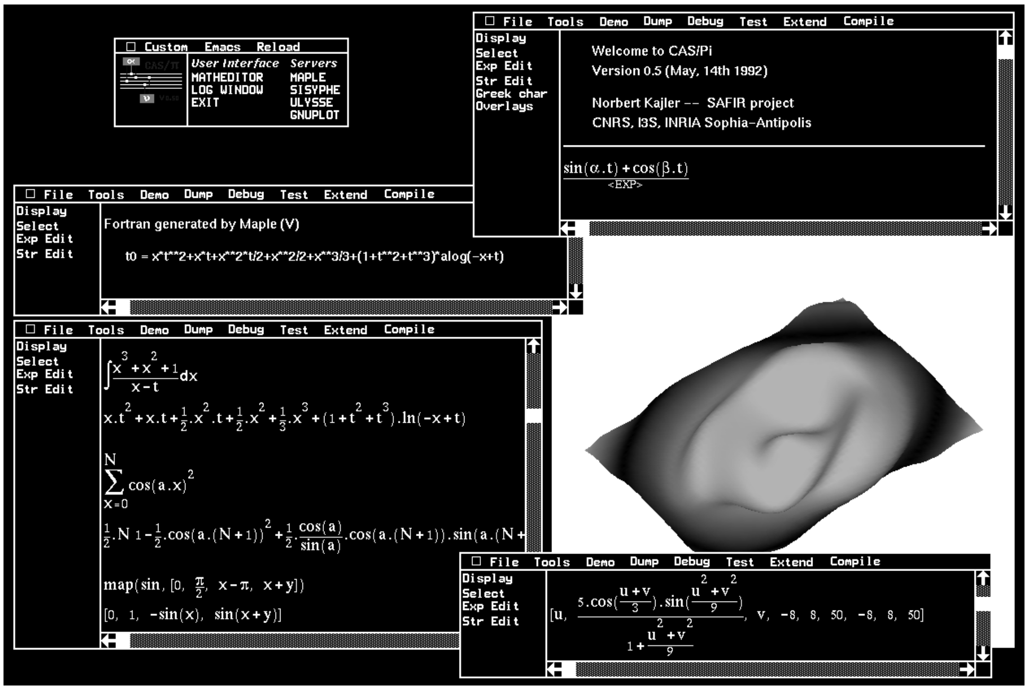

Figure 6. A sample CAS/PI session

user interface to specific needs and to connect it to various external tools. Another goal is to provide a useful workbench to ease further research in inter-systems communications and user interfaces. Possible customizations include adding new menus and panels to the user interface, new mathematical notations, new input and output data formats, and new mouse-based interactive editing operations based on the syntax and/or semantics of mathematical expressions.

Currently, CAS/PI allows concurrent access to three CASs (Maple, Sisyphe, and Ulysse) and two graphics engines (Gnuplot and IZIC). Formula editing in CAS/PI is performed using linear text, templates, or overlays. No incremental parsing is provided as in MathScribe or Theorist. However, both text parsing, template, and overlay modes may be defined and altered dynamically using CAS/PI toolkits. Figure 6 presents a sample CAS/PI session involving both Maple, Ulysse, and IZIC external servers.

The last system that we describe is MuPAD, a general purpose computer algebra system available on a wide range of hardware. MuPAD emphasizes modularity and multiprocessing(Fuchssteiner et al., 1993). In the spirit of the Axiom user interface, the MuPAD interface is based on different components running in separate windows: a textual base window to edit expressions, a debugger window, an online manual browser featuring hypertext facilities, and a graphics window to plot curves and surfaces.

3.2. NUMERICAL INTERFACES

Two systems for numerical calculations that have two-dimensional editing capabilities were developed by MathSoft Inc. The first of these, MathCAD (MathSoft, Inc., 1993), was written by Razdow in 1985 and is designed for personal computers and is limited to using their extended character set and low resolution. MathCAD allows users to mix text, equations, and plots together. They are not restricted to being on a separate line and can even overlap. MathCAD is similar to a spreadsheet in that a change to a value affects calculations and plots to the right and below the change.

MathCAD uses its own internal numerical routines and provides a limited number of operators. As a user enters an expression, it is reparsed and displayed in its two-dimensional form. Unlike other systems, expressions in numerators, denominators, exponents, etc., must be surrounded by parentheses (i.e., users should consider the underlying linear syntax). The expression is redrawn without the extra parentheses when the focus is moved outside of the expression; when the focus moves back to the expression, the expression is redrawn with the parentheses. Users can move around the expression by the use of cursor keys or a mouse and can freely edit the expression. Figure 7 show a

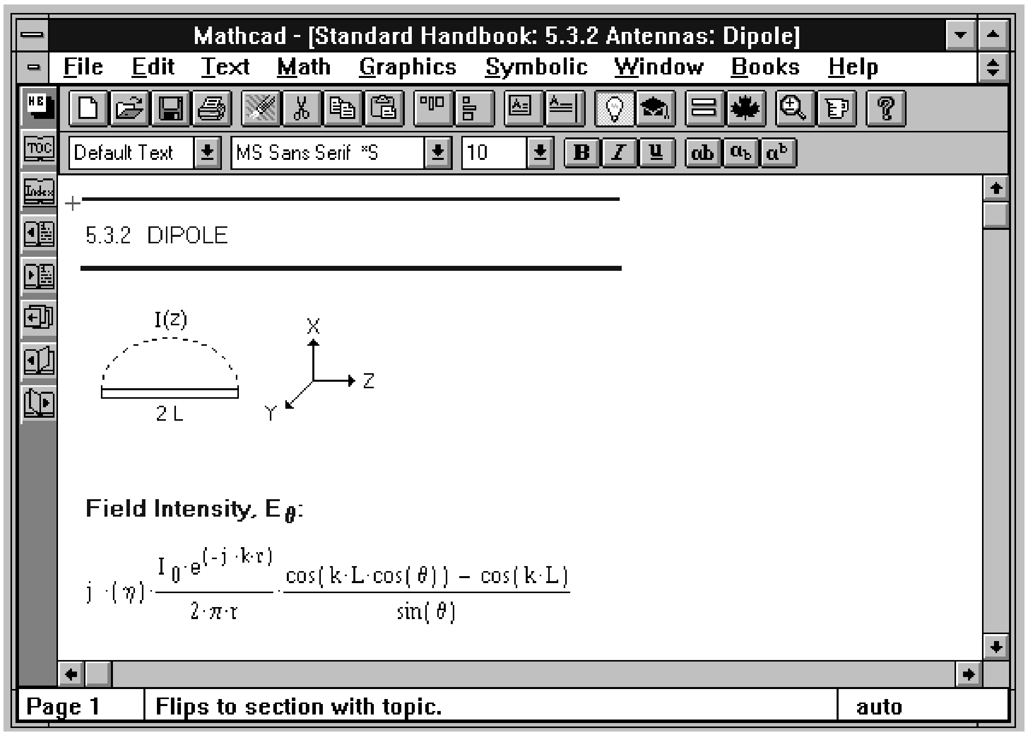

Figure 7. A sample MathCAD session

sample session.

MathCAD has been extended to fields such as hydrostatic analysis through commercially available electronic libraries. Each library includes formulas, comments, and diagrams, which may all be modified, evaluated or copied into a user's window. Some of these libraries are electronic versions of existing scientific books such as (Hicks, 1985).

Newer versions of MathCAD include a limited subset of Maple. The connection to Maple allows simple polynomial arithmetic, expansion, factoring, integration, and differentiation. It does not allow computations to be interrupted, nor does it fully integrate warning and error messages, special output forms, large return values, or line wrap the output.

MathSoft's second system is MathStation (MathSoft, Inc., 1989) and was developed by Razdow, Mueller, and Smaby in 1988. MathStation is similar to MathCAD in that it retains MathCAD's free-form mixing of text, equations, and plots and its spreadsheet metaphor. However, MathStation is designed to run on workstations and is much more sophisticated. It uses multiple fonts for text and is highly configurable by users. For example, users can define menus, key bindings, operators used, what operators look like (by writing PostScript), and how operators should be translated into FORTRAN. Users can also configure the system to use any set of graphical or numerical libraries desired, including ones the user wrote. MathStation's input model differs from MathCAD; MathStation uses the overlay model of input.

In 1990, Maple was connected to MathStation in order to add symbolic capabilities to MathStation. MathStation and Maple communicate using Maple's standard syntax. Much of Maple's syntax is the same as MathStation's syntax, but it is occasionally necessary to revert to (linear) textual mode in order to get at all of Maple's syntax and commands. Unlike CaminoReal's connections to remote algebra engines, MathStation maintains the connection throughout a MathStation session, allowing the solution of multistep problems. However, the connection does not allow a Maple command to be interrupted, nor does it support Maple debugging. MathStation is not designed to handle the intermediate and large-sized expressions that Maple can easily generate and slows down considerably when it must deal with them. MathStation does not perform automatic line-breaking, but does allow users to manually break expressions at certain points (e.g., in plus, minus, or times operators if they occur at the top level of the expression tree).

3.3. Document Processing Systems

Related to CAS interfaces are document processing systems that can typeset mathematics. In general, document processing includes three steps:

1 editing consists of defining the structure and content of the document;
2 formatting consists of setting textual and graphical elements in the plan according to various parameters such as the style of the document and the page dimensions;
3 display (printing) consists in visualizing previously formatted pages with respect to the screen (printer) resolution.

Document processoring systems can be divided into two groups: batch-oriented processors (e.g., eqn/troff (Ossanna, 1978; Kernighan & Cherry, 1978) and TEX(Knuth, 1984)) which handle these three steps sequentially and WYSIWYG editors which mix editing, formatting, and display. Batch-oriented processors suffer from two major disadvantages: their linear syntax and their lack of interactivity. To enter an expression, a user must learn the document processor's linear language. Modification must also be done in this language; the output data structures are not designed for modification. However, no existing WYSIWYG editor offers the quality of formatting achieved by TEX.

Most WYSIWYG editors do not directly support equations. Some exceptions to this are Star (Smith, Irby, Kimball, Verplank, & Harslem, 1982), Edimath (Quint, 1983; Quint, 1984), Publisher (McCarthy, Holland, & Lehman, 1987), FrameMaker 2.0 (Frame Technology, Corp., 1989), Grif (Quint, 1989), and Word (Microsfot Corporation, 1993). FrameMaker 2.0 has Milo (Avitzur, 1988) embedded in it and was described earlier. Edimath, Star, and Interleaf are similar in the way they handle expressions. They allow limited structured input and editing of expressions. An expression consists of strings (for linear subexpressions) and structures (for expressions that have vertical motion). Strings are treated like any other text. Structures are entered in a prefix manner. For example, to enter $\frac{a}{b}$, the "fraction" form is selected, the numerator is filled in with a, we move to the denominator and then fill it in with b. This unnatural method of entering infix forms in a prefix manner is mitigated by the fact that linear forms are entered as a string with no structure. Hence, expressions such as $a\,b\,+\,c$ can be entered in a natural fashion. Similarly, Word 5.0 embed a a special version of the MathType (see below) for equation editing.

VORTEX(Chen, 1988) is a TEX-based attempt at merging together the features of a batch typesetting language and a WYSIWYG editor. VORTEX does not currently support two-dimensional mathematical input, but contains hooks to support the multiple representation paradigm when such a front end is developed.

INF⊕R(Schelter, 1987) is a TEX-based WYSIWYG editor with an Emacs front-end. Because INF⊕R is written in Lisp, users can extend it by writing their own display forms in Lisp. Also, users can bind keys and use Emacs's command completion to conveniently enter mathematical notation with only a few keystrokes. In principle, INF⊕R can connect with MACSYMA or Reduce.

Several mathematics-only editors have been developed for most popular personal computers since 1985. These editors allow the typesetting of most common mathematical expressions. Once edited, typeset expressions can be copied and pasted as graphics within another application, usually a word processor. Typical of these are MacEqn (Venable, 1985) and MathWriter (Cooke & Sobel, 1986) for the Macintosh, Expressionist (Bonadio, 1987) and MathType (Design Science, Inc., 1987) for Windows and the Macintosh, and EquationBuilder (Talbot, 1992) for the NeXT computer. MathType uses palettes and presents a prefix-like, template approach although selecting an expression and command-clicking follows the overlay model. MathWriter uses palettes and menus and presents the prefix-like, template approach mentioned above. Expressionist is more like Milo in that it uses overlays. These systems are not designed to handle the large expressions produced by CAS.

More powerful document processing systems can incorporate expressions produced by mathematics-only editors. In the worst case, these expressions are treated as static images. Inter-applications protocols such as AppleEvent (Apple Computer, Inc., 1991), OpenDoc, or OLE (Microsoft Press, 1994), allow document processing systems to call external mathematics-only editors. In this way, expressions can be edited in-place within the document processing system. However, none of these protocols or representations provide a seamless interface between the document processing system and the expression editor. For example, changing the font size of the document does not cause the embedded expression to change size.

INFORM (van Egmond, Heeman, & van Vliet, 1989) is a math editor that is designed as a subsystem of a WYSIWYG document editor. Unlike the other editors mentioned, INFORM uses a parser (Heeman, 1990) based on the ideas of Kaiser and Kant (Kaiser & Kant, 1985). The parser, display, etc., are based on a grammar that is preprocessed in a manner similar to YACC (Johnson, 1978) to produce a running system. Because of the grammar, INFORM can be extended by an expert user, but not while the system is running.

This degree of extensibility is also achieved by Grif (Quint, 1989) which can also be used by expert users to generate specific structured editors from a set of specifications expressed using three languages: S (for the logical structure), P (for the layout), and T (for possible translations). In recent versions of Grif, the language S has been replaced by SGML, the Standard Generalized Markup Language, which is the ISO standard for the exchange of structured documents.

Based on Grif, the EUROMATH editor (von Sydow, 1992) extends the basic capabilities of the program by providing a Mathematical Document Type Definition (the EUROMATH DTD) and a parser/pretty-printer for LaTeX. In This way, mathematical expressions can be entered either in LaTeX format or in a WYSIWYG manner using the mouse plus some overlays palettes, with Grif providing the basic structures: texts, figures, tables, etc. Limitations include the lack of two dimensional input via the keyboard and support for handling large expressions.

The Mathematical Formula Editor (MFE) is a program by Nakayama (Nakayama, 1989). MFE is designed as a set of procedures to be incorporated into other programs (in particular, those aimed at Computer Aided Instruction), and called by them for two-dimensional input and output. MFE is similar to WYSIWYG mathematical editors in that most operators are treated as text; only two-dimensional notations have structure. One novel feature of MFE is that quotients are entered by first typing the denominator and then the numerator—the normal order of entry for Japanese mathematics. An experiment by Nakayama with 17 and 18 year-old high school girls showed that even complex formulas could be entered with only 30 minutes of training (Nakayama, 1989).

3.4. ARTIFICIAL INTELLIGENCE AND EDUCATION

Some AI work has been directed towards helping users solve problems in CASs. Genesereth (Genesereth, 1977b) discusses many of the difficulties CAS users have when trying to solve their problems. The MACSYMA Advisor (Genesereth, 1977a) is an attempt to solve some of these problems by providing an interactive "consultant." Gardin and Campbell (Gardin & Campbell, 1981; Gardin & Campbell, 1983) present a system for Reduce that attempts to help inexperienced users correct common "mistakes" when dealing with CASs (e.g., removal of unnecessary evaluations for procedure arguments and use of global flags). More recently, a conference dedicated to Artificial Intelligence and Symbolic Mathematical Computing took place in Karlsruhe, Germany. Two papers (Butler, 1992; Calmet & Campbell, 1992) reference previous works in this area and investigate future research directions.

Another related direction has been the use of CASs in education. To date, the use of CASs in classrooms has been limited, but is growing rapidly; (Buchberger, 1985; Steen, 1988; Karian, 1992) discuss how CAS are used and might be used in education at both the high school and college levels from both the computer algebra community's and educator's point of view. (Bauldry & Fiedler, 1990) and (Skiena, 1990) are among a number of texts that teach various fields of mathematics with the assumption that a CAS is being used. (Brown, Porta, & Uhl, 1991) and (Porter & Hill, 1994) are two examples of interactive mathematics texts (written respectively with Mathematica and MathCAD) used in classrooms. (Scheftic, 1993) lists some guidelines to build interactive mathematics texts using a CAS. User interfaces specifically designed for educational purposes include Newton (Lamagna, Hayden, & Johnson, 1993) and Calculus T/L II (Child, 1993).

Some CASs specifically designed for teaching are EQD (Ager, Ravaglia, & Dooley, 1989; Suppes, Ager, Berg, Chuaqui, Graham, Maas, & Takahashi, 1987), MATHPERT (Beeson, 1989), and Bunny Numerics (Graci, Narayan, & Odendahl, 1989). Student (Devitt, 1989) is a Maple package designed to turn a CAS into a more pedagogically satisfying tool. Among the issues addressed by these packages, the most important in terms of user interaction are the correctness during the course of a computation, the ability to provide explanations, and the possibility to solve problems step by step according to the student's level.

With regard to education, a simple user friendly front end is certainly important. Also important in terms of human interaction are the use of standard mathematical notation, explanation of results, and guidance for the user as to what might be useful to try next. Lastly, highly interactive front ends that allow for experimentation, in the spirit of the Avitzur's Graphing Calculator (Avitzur, 1995) (see figure 8), should make mathematics more attractive to students.

4. The Problem

4.1. ENTERING EXPRESSIONS

In traditional CASs, expressions are entered in a conventional linear syntax. The lack of two-dimensional input leads to both syntactic errors (missing commas, parentheses, etc.) and structural errors (wrong expression, missing subexpression, etc.). It forces users to learn and use an unfamiliar and possibly clumsy notation. Traditional CASs parse input only after a complete command has been typed; most errors are not discovered until after the expression/command has been completely typed. No CAS attempts automatic error correction and only a few systems allow users to edit commands (usually with a line-oriented editor which has its own syntax to learn).

Some of the modern interfaces use two-dimensional input. Most of these interfaces use either templates or overlays that are chosen from menus or palettes (menus that are permanently on the screen) or from their keyboard equivalents. Entering expressions using templates and overlays is data-driven and is inherently extensible (i.e., the algorithm is independent of the template/overlay that is used). Overlays allow infix expressions to be entered in a natural left-to-right order, an advantage over templates. However, templates and overlays do not allow natural left-to-right input of a number of mathematical notations (e.g., integration) and programming language constructs†. Parsing allows natural left-to-right input for all of these cases. However, using a parser may require users to type parenthesis that are not normally used in the display of the expression. For example, the expression $\frac{x-1}{x+1}$ might be entered as `(x-1)/(x+1)` using a parser; the parentheses surrounding the numerator and denominator are not used in the display of the expression.

† Most CASs have a programming language embedded in them.

Also, parsers must tolerate incomplete syntactic forms that occur before an expression is fully entered.

Some traditional CASs, such as MACSYMA (MATHLAB Group, 1977) and SMP (Cole & Wolfram, 1981), have extensible parsers that allow users to introduce new syntax (operators). However, the syntax extensions are restricted to be either prefix, postfix, infix, or matchfix (e.g., brackets) operators; other forms of syntactic extension such as programming constructs are not allowed. Most parsing algorithms are not extensible.

All of the modern interfaces mentioned in Section 3 enforce correct syntax at all times. The mathematical editors used in document processing systems treat most input as linear strings—"syntax" is limited to two-dimensional forms such as quotients, subscripts, and superscripts. Syntactic correctness can be a nuisance, particularly during editing, and is one of the reasons structure editors such as the Cornell Program Synthesizer (Teitelbaum & Reps, 1981) and Gandalf (Notkin, 1985) have not been widely accepted for programming languages.

4.2. Selecting and Editing Expressions

Commands to CASs often use parts of previous commands or previous output. *Selection* is the process of "grabbing" some subexpression of an expression. Traditional CASs perform selection through a few specialized commands such as `numerator` and `rhs` and through a general command (usually called `part`) which requires users to mentally walk through the underlying parse tree which can differ from the displayed expression. Command-based selection is both error-prone and unnatural.

Modern interfaces use a mouse to select subexpressions, although selection techniques vary. Using a mouse for selection is essentially error-free because immediate feedback can be provided. Those interfaces that require syntactic correctness allow only syntactically correct subexpressions to be selected; most of the document processing interfaces allow more general selection because most of the expression is treated as a string. For structured notations, selection of multiple subexpressions that do not correspond to the internal representation remains problematic (e.g., selecting all diagonal entries in a matrix). Non-structural selection can also be provided by circling (using a pen), as shown in (Genesereth, 1979).

CASs often rearrange expressions in order to more efficiently manipulate them. The resulting expression is often not in the form that users desire or expect (Moses, 1971). Some traditional CASs allow in-place editing of expressions in order to manipulate subexpressions into the desired form (e.g., a factored denominator). However, just as with subexpression selection, users must mentally walk through the underlying parse tree.

4.3. Direct Manipulation

Another interesting area concerns direct manipulation of mathematical expressions involving both syntactic and semantic aspects. A related (and desirable) goal consists of mixing the editing and simplification processes, so that, for instance, moving a symbol in an equation implicitly means "solve the equation with respect to this variable". As shown in the previous section, Milo or Theorist already achieve this to some extent. Using Theorist, the user can, for instance, move subexpressions around while the system maintains the meaning of the surrounding expression or makes substitutions by dragging and dropping formulas. However, more work could be done in this direction. First, the direct manipulation approach could be extended in many directions to support all of the usual operations: factorizing, expanding, performing variable substitutions, solving equations, shrinking/zooming subexpressions, etc. Second, editing and simplification could be coupled in such a way that the editor would automatically enforce a presentation style based on some rules specified by the user. In this way, the editor could systematically try to isolate a variable or order elements in a sum or a product after every computation, and also after every interactive manipulation.

Mathematical assistants like Milo or Theorist achieve direct manipulation by mixing the editing and the simplification process within a single software component. This works very well with a single application, but other problems arise in the case of a collection of independent symbolic computation packages used from a single editor: which component is in charge of the simplification in relation to the editing process? How do these two components communicate?

4.4. FORMATTING AND DISPLAYING EXPRESSIONS

Most traditional CASs display expressions in their natural two-dimensional form within the limits of a character terminal (Figure 1). Modern interfaces use bitmapped displays. Some early attempts used eqn (Kernighan & Cherry, 1978)/troff (Ossanna, 1978) to improve the quality of the display of expressions. There are several problems with using batch-oriented word processors such as eqn/troff and TEX(Knuth, 1984) to display expressions from a CAS. One problem is that large expressions run off the edge of the screen and there is no way to recover the lost information. Another problem is that troff and TEX simply "spray" the bits of an expression onto the screen without retaining any of its internal structure, so there is no way to interact with the output. Lastly, the fonts that are used are a problem. Traditional mathematical typesetting uses many different sizes of normal, italic, and special fonts and were designed to be used with output devices capable of printing 300 or more dots per inch. The resolution of the display on a workstation is typically between 65 and 100 dots per inch†. At 65 to 100 dots per inch and a 10–12 point base font, Eqn and TEX's output looks poor—italic fonts are especially hard to read and nested superscripts are unreadable if they are reduced in size as is done in traditional typesetting. The only way to compensate for this is to either abandon traditional typesetting practices and only reduce subscripts and superscripts slightly, if at all, or to use larger-sized base fonts. The drawback to the latter solution is that only small expressions will fit on the workstation screen.

CASs frequently generate large expressions. In fact, their ability to manipulate them is one of the main reasons for using a CAS. It is therefore surprising that most of the interfaces for CASs handle large expressions poorly. In general, large expressions are treated no differently than small expressions except that they are broken up and displayed over several lines‡. Each system expends a different amount of effort in determining where a line break should occur. Because of line breaks, operators that have "vertical

† Higher resolution screens are available, but they remain very expensive. The focus of this paper is on software ideas that can be used on commonly available hardware today and in the near future.

‡ A number of systems have some special printing functions. For example, Mathematica (Wolfram, 1988) has a function `Short` that prints an expression using a specified number of lines by only showing the first and last parts of the expression and eliding internal parts. MACSYMA (MATHLAB Group, 1977) has a function `printpois` that can be used to print (large) Poisson series expressions in a more readable format.

motion" (such as quotient) that span more than one line must be reformatted into a linear format. This makes it harder to comprehend an expression whose large size already makes comprehension difficult. The introduction of line breaks cause very large expressions to scroll off the top of the screen.

A few modern interfaces allow horizontal and/or vertical scrolling of expressions, elision (because of either breadth or depth), and renaming of subexpressions. The number of terms elided or other descriptive information is sometimes indicated. In CAS/PI, the handling of large expressions is largely customizable: users can set a series of parameters which limit the display of large expressions, and functions can be defined to specify how elided sub-expressions should be displayed. Miniature fonts have also been tried; they might work well in combination with the notion of fish-eye lenses (Burke & Fisher, 1987).

Another problem with large expressions is that they take a long time to display. There are two reasons for this.

1 Most systems are tree-based and format each subexpression in isolation from other subexpressions. However, large expressions often have subexpressions that occur repeatedly within them, so the display of a common subexpression is (re)computed for each occurrence of the subexpression. If line breaking is used, recomputation may be necessary if the same subexpression must be broken across lines and, hence, displayed differently.
2 The introduction of a line break can cause the display algorithm to backtrack (because of vertical motion operators) and recompute the position of the subexpression.

Most interfaces do not allow users to specify new display forms for new operators. Of the traditional CAS interfaces, only Axiom and Mathematica allows users to specify display form for new operators, but the specification is limited to ASCII strings. Both MathStation and CAS/PI allow users to specify some kinds of new graphical display forms, either using PostScript (MathStation) or a specific 2D pretty-printing meta language (CAS/PI). In both cases, such adaptation of the user interface is meant for expert users.

4.5. Ambiguous notation

There are two ways in which notation can be ambiguous. Generic notation uses the same notation to represent similar but different functions. For example, "+" is used to mean polynomial addition, matrix addition, etc. This is usually not a problem for CASs except when there are major semantic differences in meanings such as is the case for multiplication which is commutative for polynomials but noncommutative for matrices. Most CASs "solve" this problem by introducing a different operator for noncommutative multiplication. For domain-oriented CASs such as Axiom (Jenks & Sutor, 1992), this is not a problem.

The second and more fundamental problem is that different fields of mathematics and engineering use the same notation in different ways. Understanding a notation requires knowledge of the problem domain. For example, f' means "first derivative" in calculus and analysis and means 'a variable different from f' in other domains. Other examples include $\bar{x}$ (conjugation, mean, negation) and i (integer, $\sqrt{-1}$). Conversely, different notations are used to mean the same thing. For example, $\sqrt{-1}$ is usually represented by i in mathematics but by j in electrical engineering.

More details concerning typesetting of mathematical notations can be found in the various guidelines available to authors from most scientific book and review publishers. From a different perspective, Cajori published a history of mathematical notations in two volumes which covers both elementary notations and more complex ones (Cajori, 1974). More recently, there has been some work at IBM Watson Research Center directed towards using mathematical notations as a programming language. Directly related to mathematical notations, two papers were published: a study of ordinary mathematical notations, highlighting some ambiguities and their implications for editing, interpreting, and compiling (Driscoll, 1990), and a context-free grammar (Revesz & Lynch, 1991). Lastly, Leslie Lamport proposed the introduction of some specific notations to deal with large mathematical expressions (one or two pages large in size) (Lamport, 1993).

4.6. SESSION LAYOUT

In traditional CASs, input and output are mingled in a single window and scroll off the top of the window never to be seen again unless some provisions are made for preserving a record (e.g., by storing an expression in a variable or by support for scrolling in a terminal emulator). Many modern interfaces allow multiple windows and expressions to be added or deleted at any place in a window. Adding, deleting, or editing an expression that is not at the bottom of the window can confuse what appears to be a linear flow from the top of the window to the bottom of the window. On the other hand, the ability to delete unimportant results (diagnostic output, part selection, etc.) can clarify derivations and free valuable screen space.

PowerMath (Davenport & Roth, 1986) avoids the linear flow problem by putting each expression into its own window. The result of a computation depends upon which windows are opened at the time of the computation. This can be confusing in the same way that adding or deleting an expression in the middle of a window can be confusing—there is not necessarily a connection between the assumptions and the results. Another problem with putting each expression in its own window is that very few expressions can be "remembered" because window clutter soon takes over the screen.

In many interfaces, the scope of a variable is not limited to the window in which it is used. This can cause confusion because dependency information, such as assignment ordering, is not obvious across windows. Both GI/S (Young & Wang, 1987) and Theorist (Bonadio, 1989) limit the scope of a variable to the window in which it is used. Theorist also maintains dependency information.

4.7. GRAPHICS

Most CAS include a "captive" graphic engine to plot curves and/or surfaces. These graphics routines were written to work specifically with a given CAS. This is the case with Axiom, Derive, Maple, Mathematica, MuPad, and Theorist. Some plotting engines were developed inside the Computer Algebra community, but separately of any CAS. This includes: SIG (Wang, 1990), IZIC (Fournier, Kajler, & Mourrain, 1993; Kajler, 1994), and the plotting engine included in MathScribe (Tektronix, Inc., 1988).

More generally, it is possible to connect any specialized plotting engine to a CAS by defining one or more functions to compute a set of points from a list of equations and ranges, and use this set of points as an input of the graphics engine. This method is used by SIG and IZIC. It has also been used with Mathematica to connect to AVS (Upson, Faulhaber, Kamins, Laidlaw, Schlegel, Vroom, Gurwitz, & van Dam, 1989) and to IRIS Explorer (Edwards, 1992). However such a one-way communication between the CAS and the plotting engine limits interactivity (Avitzur, Bachmann, & Kajler, 1995).

In 1994, Ron Avitzur presented the Graphing Calculator (Avitzur, 1995), a highly interactive graphing tool shipped with every Power Macintosh. Figure 8 shows an example

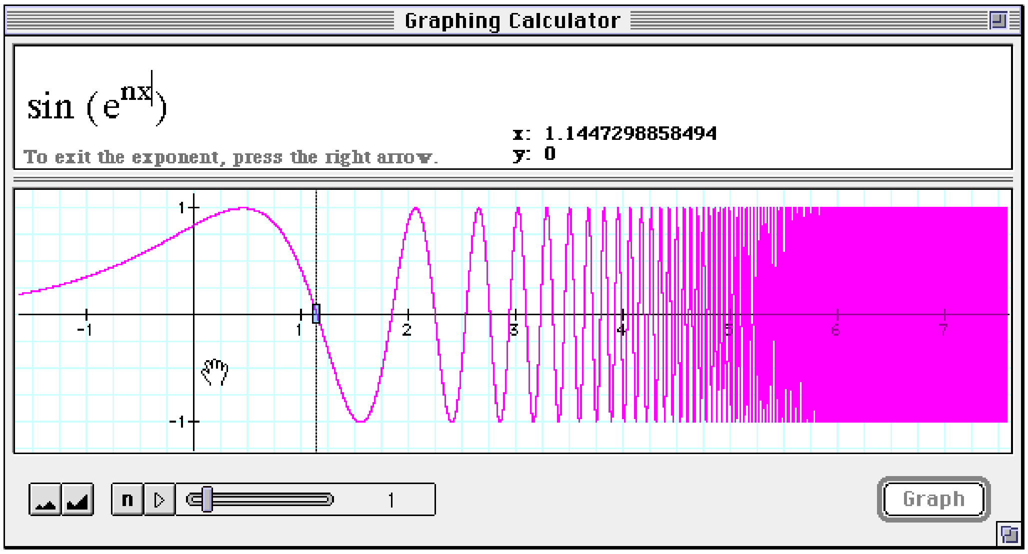

Figure 8. The Graphing Calculator

of the use of the Graphing Calculator. The graph was produced with the "honest plotting" capability (see below) turned on. A slider is used to interactively animate the curve according to the parameter n. Roots are numerically computed by pointing at them with the mouse. While zooming or unzooming, the software computes more points and updates the display in real time.

Also, (Fateman, 1992) investigates the use of interval arithmetic to improve the correctness of curve plotting. Fateman referes to this method as "honest plotting" as it systematically provides a rigorous – but sometimes pessimistic – bound to the place where the curve actually lies. In the same spirit, (Avitzur, Bachmann, & Kajler, 1995) investigates a collection of techniques for curve and surface plotting which they call "intelligent plotting". Intelligent plotting relies on the transparent use of numeric and symbolic methods to improve the correctness, efficiency, and user-friendliness of plotting packages.

4.8. SOFTWARE ENGINEERING ISSUES

4.8.1. CONNECTION BETWEEN THE USER INTERFACE AND THE KERNEL

In most CASs, the user interface is an internal component of the CAS. Typically, these systems include parsing and display procedures that are called by some top level loop code. This is the case for Macsyma, Reduce, Derive, Milo, and Theorist. However, decoupling the user interface from the algebraic kernel offers many advantages:

1 separate development and updating of the two programs;
2 substitution of the whole user interface;
3 running the user interface and the kernel on different computers;
4 simultaneous access to different engines from a common user interface.

Three commercially available CASs feature a separate user interface: Axiom, Maple, and Mathematica. In addition, GI/S (Young & Wang, 1987), MathScribe (Smith & Soiffer, 1986), and MathStation (MathSoft, Inc., 1989) may be considered as independent user interface for an existing CAS (resp. Macsyma, Reduce, and Maple); while CaminoReal (Arnon, Beach, McIsaac, & Waldspurger, 1988), CAS/PI (Kajler, 1992b), and SUI (Doleh & Wang, 1990) are independent user interfaces allowing simultaneous use of different remotely connected CAS. In the following, we sketch the communication protocols used by MathScribe, CaminoReal, Maple, Mathematica, and CAS/PI.

MathScribe

While the separation between the user interface and the computational engine (i.e. Reduce) is strictly enforced in MathScribe, both were compiled as a single process for efficiency reasons. MathScribe includes a specific software component to isolate the computational engine specificities from the user interface and to manage input and output dataflows by separating requests, answers, error messages and questions from the engine.

In order to test the portability of MathScribe, an experimental version was later developed with Maple in place of Reduce. In this second version, Maple and MathScribe were two processes which communicated via sockets.

CaminoReal

CaminoReal communicated with external CASs in their own language using command strings. CaminoReal converted its internal data structures into the appropriate strings for the external CAS. Results were returned as linear expressions and parsed into CaminoReal's internal structures. Thus, a parser and unparser were written for every CAS to which CaminoReal was connected.

Maple

Beginning with version V, Maple has been composed of a kernel and a set of devices, including a user interface, Iris, and a plotting engine. The kernel and devices can run on remote computers. The communication follows a specific protocol (Leong, 1986). More precisely, data can be passed in one of two ways:

1 Strings can be exchanged, where the contents of the string are suitable to be used as input to Maple.
2 Data can be exchanged as Directed Acyclic Graphs (DAG) using the internal data representation of Maple. When necessary, the DAGs are invisibly encoded as ASCII strings. Using DAGs has two advantages: first, DAGs can reduce the amount of data transmitted by sharing common subexpressions; second, using Maple's internal data representation eases data encoding and decoding. Routines are provided to simplify construction of, and access to the DAGs.

Until 1994, this division was only accessible with a special OEM version of Maple. Recently, Maple introduced MathEdge (Pintur, 1994), a development toolkit which enables applications developpers to link their applications with the kernel of Maple V.

Mathematica

Beginning with version 2, communication to Mathematica is done using MathLink (Wolfram Research, Inc., 1993a). MathLink implements a communication protocol and provides a set of procedural interfaces that allow C programs to send and receive data, to call (or be called) by Mathematica, or even different instances ot Mathematica to communicate with each other. MathLink is fully documented and library routines are provided so that advanced users can use it for their own applications. MathLink's interface exposes Mathematica's representation of expressions, although `ToString` and `ToExpression` can be used so that strings are sent instead of the internal structure. Because the details of the communication are hidden, MathLink could be used to transmit DAGs. However, version 2.2 does not have this optimization.

Independent of MathLink, a commercial package named InterCall (Robb, 1992) allows C, Fortran, and Pascal programs to communicate with Mathematica. Also, commercially available software such as Excel and LabView have been hooked up to Mathematica so that they act as front ends to it.

CAS/PI

Like CaminoReal, CAS/PI's user interface components and CASs communicate using any convenient concrete language which is parsed and unparsed on CAS/PI's side. However, CAS/PI's kernel also comes with a software bus which allows plugging and unplugging of software components as well as easy programming of inter-component communications (Kajler, 1992a). In this way, each software tool (e.g. formula editor, panel, remote CAS, other external application) communicate by sending and receiving typed messages, a type being the name of a service possibly provided by one or more tools plugged on the same bus at that time. According to its type, each message is broadcasted by the bus to all other tools which have expressed interest in that type of message. Additional mechanisms provided by the CAS/PI protocol can be used to limit the broadcasting of a particular message to some specific tools and/or to specify a continuation, e.g. the collection of tools which should receive the answer to the message. As the whole broadcasting is asynchronous and based on the instantaneous state of the bus, concurrency and runtime extensibility are side-effects of this approach. For the user, this means that it is possible to keep editing expressions while one or more computations are handled concurrently by remote CASs. However, this capability is limited by some missing features such as a graphical feedback of the set of on-going computations and some easy way to send interruptions.

4.8.2. Portability to different CASs

Every CAS has its own (limited) user interface. Portability of the interface to other CASs is not really a problem with today's CAS interfaces because they are tied to a particular CAS. However, a good interface requires a substantial amount of work and this work should not have to be duplicated for each CAS. Therefore, it is desirable to produce a portable interface that handles lexical, syntactical, and functional differences between different CASs.

There are two extremes to a portable CAS interface design: fully expose the underlying algebra system (both its strengths and its weaknesses) or try to hide the computer algebra system by defining a new syntax and set of functions. The latter approach requires writing numerous procedures for each algebra system to present a similar interface. For example, sending the simple expression $a + b/c$ to Reduce returns the answer $(ac + b)/c$ whereas Maple returns the original expression. The differences in canonical forms means that even the "plus" operation must be dealt with for each CAS. A middle ground is to hide lexical and syntactic differences; function names, argument order, and function results would be different for each CAS except for those functions which have a common mathematical notation (e.g., $+$ and $\int$).

Currently, several on-going projects within the computer algebra community are seeking to establish a standardized protocol for the exchange of mathematical expressions between applications. These projects include Multi (Gray, Kajler, & Wang, 1994), OpenMath (OpenMath Group, 1994), and PoSSo (Gonzalez-Vega, L. and Recio, T., editors, 1994).

4.8.3. Extensibility of the user interface

It is important that solutions to the above problems be extensible by users to handle new notations. Notation is used to convey information succinctly to the problem solver and mathematicians develop new notation when existing notation is clumsy or non-existent for the problem being solved. If an interface cannot handle a new notation, then it hinders problem-solving. Also, user interface components such as menus, panels, etc., should be configurable by the user to fit his specific needs. Moreover, loading a new package in a CAS should update the user interface to give a easy and immediate access to the added functionalities. More generally, programmability of the user interface empowers users and third party developers to tailor the interface to specific needs and wishes. This approach was validated a long time ago by Emacs (Stallman, 1979) in the domain of text editors and more recently by Tk (Ousterhout, 1994) in the field of graphical user interfaces. In this spirit, CAS/PI provides a collection of four Lisp toolkits that allow runtime extensibility of different aspects of the system: the set of mathematical notations, the menu-panels user interface, the mouse-based editing functionalities, and the software bus architecture (Kajler, 1993b).

4.9. Efficiency considerations

It is important that any solution to these problems be space and time efficient because of the large size of expressions that CASs can generate. To date, all interfaces for two-dimensional editing (with the exception of MathScribe), use trees and not DAGs as their fundamental data structure and are less efficient in their use of storage for large expressions. Trees require an amount of space that is linearly proportional to the number of nodes displayed. By contrast, DAGs only require an amount of space proportional to the number of unique nodes displayed. Experimental results in (Soiffer, 1991) show that DAGs use between 5% and 15% of the space used by trees for some sample sessions. For a Reduce demonstration script, this results in a savings of 540kb memory, assuming 40 bytes/node (subexpression)†.

Another aspect of efficiency is incrementality. This relates to both editing and exchange of large mathematical expressions. When editing large expressions, it is essential that updates to the display of mathematical expressions are made incrementally. This means that when a subpart of an expression is modified, recomputing of layout and redisplay should be limited exclusively to the relevant subexpression. When evaluating an expression, exchange of large expressions could be optimized by only sending back and forth parts of expressions that will really be used by the remote program. In the case of a mathematical editor using auto-elision or some similar mechanisms to deal with large expressions, the remote engine should limit its transfer to the subexpressions which will actually be visible on the screen according to the current option setting, dimensions of windows, etc. To date, only CAS/PI includes such a lazy way to send only relevant subexpressions from computer algebra systems to the user interface.

4.10. Alternative Input Technologies

The focus of this paper has been on what Wells (Wells, 1972) calls *keyboard languages.* These languages have the property that characters are placed on the screen by special keys and that the placement of the characters is controlled by these keys. In contrast to this approach, *pen languages* allow the user to draw characters on the screen in arbitrary positions and, through the use of handwriting recognition systems, interpret what characters have been drawn and their meaning (e.g., exponent, etc.). Pen languages have the obvious appeal that commands are not needed to indicate the two dimensional structure of an expression. However, the extremely large character set used in mathematics, together with the lack of context that words provide, tends to reduce the accuracy of character recognition algorithms for mathematics.

The remainder of this section briefly discusses alternatives to using the keyboard and mouse.

4.10.1. Handwriting Recognition

A pen is a very powerful device allowing:

1 input of data using handwriting recognition,
2 pointing,
3 initiation of actions through gesture recognition.

For most applications, a pen-based user interface may reduce learning time and ease access to the computer by unifying input devices (i.e., mouse and keyboard). However, keyboards remains the most efficient device for purely textual data input (Brown, 1988).

Early work on recognizing handwritten expressions was done by Anderson (Anderson, 1968) and by Martin (Martin, 1971). Both Anderson's and Martin's algorithms worked by reducing the recognition problem to a parsing problem by linearizing the input. Unfortunately, linearization imposes some constraints on what can be recognized. Anderson presents a slower algorithm that does not have these constraints.

† The amount of space used per node varies considerably depending the system. 40 bytes per node is on the low end.

More recently, work at IBM by Orth (Orth, 1990) has focused on building up relations such as "near to" and "to the right of" in considering placement of characters, and using these relationships to determine what is a superscript, etc.

In 1991, Marzinkewitsch presented a prototype using a neural network to recognize hand-written expressions (Marzinkewitsch, 1991). The system also performed symbolic computations by translating expressions back and forth to Maple and Reduce systems.

In 1992, Avitzur (Avitzur, 1992) showed a prototype running on a Macintosh. This prototype recognizes input of mathematical expressions with a custom built character recognizer. Avitzur's recognizer has been able to distinguish between as many as 200 different characters, but to do so, a significant amount of training time was required. Avitzur's program uses gestures for deleting (a rubbing out gesture) and selecting (circling). More complicated gestures to factor, expand, etc., were found to be confusing to users as there was no intuitive basis for them. The program also included the direct manipulation features of Milo (Avitzur, 1988), an earlier program by Avitzur.

4.10.2. Speech recognition

Another input form is voice. Although speech recognition is an area of active research, we are not aware of any implementation that uses speech recognition to allow entry of expressions. This is unfortunate because speech recognition (and synthesis) may allow disabled people easier access to scientific software. The *Handbook for Spoken Mathematics* written by Chang in 1983 (Chang, 1983) may be of use for those who wish to pursue this area. The handbook specifies how to verbally express common mathematical expressions in order to achieve clarity and unambiguity.

It is important to note that voice input must eventually be interpreted as commands. As such, voice input is really just another form of "keyboard" input and can be layered on top of an existing system.

Acknowledgement

Many people contributed to this survey, some of them directly during the writing of this paper in 1993, some others indirectly by providing materials to Kajler's and/or Soiffer's PhD thesis before that. We would like to thank all contributors here, with special mention to Dennis Arnon, Ron Avitzur, Allan Bonadio, Bruce Char, Sam Dooley, Richard Fateman, André Galligo, Simon Gray, Theodore Gray, Jerry Porter, Doug Stein, and Paul S. Wang.

References

Ager, T. A., Ravaglia, R. A., Dooley, S. (1989). Representation of Inference in Computer Algebra Systems with Applications to Intelligent Tutoring. In E. Kaltofen & S. M. Watt (Eds.), *Computers and Mathematics*, pp. 215–227. Springer-Verlag.

Anderson, R. H. (1968, January). *Syntax-Directed Recognition of Hand-Printed Two-dimensional Mathematics*. Ph. D. thesis, Harvard University, Cambridge, MA. Shorter version in M. Klerer and J. Reinfelds (eds), *Interactive Systems for Experimental Applied Mathematics*, pages 436–459, 1968, Academics Press.

Anderson, R. J. (1983). EXED: A Prototype Environment for Algebraic Computation. Unpublished Manuscript, Computer Science Dept, UC Berkeley.

Antweiler, W., Strotmann, A., Winkelmann, V. (1989, April). A TEX-REDUCE-Interface. *ACM SIGSAM Bulletin* **23**(2), 26–33.

Apple Computer, Inc. (1991). *Inside Macintosh.* New York: Addison Wesley.

Arnon, D. (1987, November). REPORT of the Workshop on Environments for Computational Mathematics, held July 30, 1987, during the ACM SIGGRAPH Conference. *ACM SIGSAM Bulletin* **21**(4), 42–48.

Arnon, D., Beach, R., McIsaac, K., Waldspurger, C. (1988). CaminoReal: An Interactive Mathematical Notebook. In J. C. van Vliet (Ed.), *Proc. EP'88 International Conference on Electronic Publishing, Document Manipulation, and Typography*, Nice, France, pp. 1–18. Cambridge University Press. Also available as Technical Report EDL-89-1, Xerox PARC, 1989.

Arnon, D., Waldspurger, C., McIsaac, K. (1987). CaminoReal User Manual Version 1.0. Technical Report CSL-87-5, Xerox PARC.

Avitzur, R. (1988). *Milo* (a Macintosh computer program). Paracomp Inc. San Francisco, CA. Milo has been incorporated in FrameMaker, Frame Technology Corp. San Jose CA.

Avitzur, R. (1992, April). Your own handprinting recognition engine. *Dr Dobbs Journal*, 32–37.

Avitzur, R. (1995). Direct Manipulation in a Mathematics User Interface. In N. Kajler (Ed.), *Human Interaction for Symbolic Computation*, Texts and Monographs in Symbolic Computation. Springer-Verlag.

Avitzur, R., Bachmann, O., Kajler, N. (1995, January). From Honest to Intelligent Plotting. Technical Report 5, RIACA, Amsterdam. Submitted to ISSAC'95.

Bauldry, Fiedler (1990). *Calculus Laboratories With Maple.* Brooks/Cole.

Beeson, M. J. (1989). Logic and Computation in MATHPERT: An Expert System for Learning Mathematics. In E. Kaltofen & S. M. Watt (Eds.), *Computers and Mathematics*, pp. 202–214. Springer-Verlag.

Bonadio, A. (1987). *Expressionist* (a computer program). 939 Howard St., San Francisco, CA, 94103: Prescience Corp.

Bonadio, A. (1989). *Theorist* (a computer program). 939 Howard St., San Francisco, CA, 94103: Prescience Corp.

Brown, C. M. (1988). Comparison of typing and handwriting in "two finger" typists. In *Proc. of the 32nd Annual Meeting of the Human Factors Society*, pp. 381–385.

Brown, D., Porta, H., Uhl, J. J. (1991). Calculus and *Mathematica*: A Laboratory Course for Learning by Doing. In L. C. Leinbach, J. R. Hindhausen, A. M. Ostebee, L. J. Senechal, & D. B. Small (Eds.), *The Laboratory Approach to Teaching Calculus*, Volume 20 of *MAA Notes*. Washington, D.C.: The Mathematical Association of America.

Buchberger, B. (Ed.) (1985). *Special Session on Symbolic Mathematical Systems and Their Effects on the Curriculum.* Available as ACM SIGSAM Bulletin, 18/19(4/1):3–62.

Burke, M. G., Fisher, G. A. (1987, April). A Practical Method for LR and LL Syntactic Error Diagnosis and Recovery. *ACM Trans. On Programming Languages and Systems* **9**(2), 164–197.

Butler, G. (1992, August). The Progress Towards an Intelligent Assistant — A Discussion Paper. In J. Calmet & J. A. Campbell (Eds.), *Proc. of Artificial Intelligence and Symbolic Mathematical Computing*, Volume 737 of *LNCS*, Karlsruhe, Germany, pp. 107–115. Springer-Verlag.

Cajori, F. (1952–1974). *A History of Mathematical Notations (two volumes).* Open Court.

Calmet, J., Campbell, J. A. (1992, August). Artificial Intelligence and Symbolic Mathematical Computations. In J. Calmet & J. A. Campbell (Eds.), *Proc. of Artificial Intelligence and Symbolic Mathematical Computing*, Volume 737 of *LNCS*, Karlsruhe, Germany, pp. 1–19. Springer-Verlag.

Chang, L. A. (1983). Handbook for Spoken Mathematics. Technical report, Lawrence Livermore Laboratory.

Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., Watt, S. M. (1991). *Maple V Language Reference Manual.* Springer-Verlag. ISBN 0-387-97622-1.

Chen, P. (1988). *A Multiple Representation Paradigm for Document Development.* Ph. D. thesis, EECS Dept, University of California, Berkeley, California. Report No. UCB/CSD 88/436.

Child, D. (1993). *Calculus T/L II: a Program for Doing and Learning Mathematics.* Brooks Cole. Demonstration package.

Clapp, L. C. (1986, April). Magic paper: An On-Line System for the Manipulation of Symbolic Mathematics. Technical Report R 105-1, Computer Research Corp., Newton, Mass.

Clapp, L. C., Kain, R. Y. (1963, November). A Computer Aid for Symbolic Mathematics. In *Proc. AFIPS Fall Joint Computer Conference*, Volume 24, pp. 509–517. AFIPS Press.

Cole, C., Wolfram, S. (1981). *SMP Handbook.* Caltech.

Cooke, J. R., Sobel, E. T. (1986). *MathWriter* (a Macintosh computer program). Ithaca, New York: Cooke Publications.

Davenport, J. H., Roth, C. E. (1986, July). PowerMath: A system for the MacIntosh. In *Proc. 1986 Symposium on Symbolic and Algebraic Computation*, pp. 13–26. ACM Press.

Design Science, Inc. (1987). *MathType User Manual* (a computer program). 4028 Broadway, Long Beach, CA 90803.

Devitt, J. S. (1989, July). Unleashing Computer Algebra on the Mathematics Curriculum. In *Proceedings of ISSAC'89*, Portland, Oregon. ACM Press.

Doleh, Y., Wang, P. S. (1990, August). SUI: A system Independent User Interface for an Integrated Scientific Computing Environment. In *ACM Proc. of the International Syposium on Symbolic and Algebraic Computation*, Tokyo, pp. 88–94. Addison-Wesley.

Driscoll, G. C. (1990, October). Ordinary Mathematical Notation: Some Characteristics and their Implications for Programming. Technical Report RC 15365, IBM Research Division, T.J. Watson Research Center.

Edwards, G. (1992). *Image Processing*, Chapter Visualization – the Second Generation, pp. 48–53.

Fateman, R. (1987, November). TEX Output from MACSYMA-like Systems. *ACM SIGSAM Bulletin* **21**(4), 1–5.

Fateman, R. (1992, July). Honest plotting, global extrema and interval arithmetic. In P. S. Wang (Ed.), *Proc. of ISSAC'92*, Berkeley, USA, pp. 216–223. ACM Press.

Fateman, R. J. (1979). MAC-ED, an Interactive Expression Editor for MACSYMA. In *Proc. 1979 MACSYMA Users' Conference*, pp. 336–343.

Foderaro, J. K. (1978, December). Typesetting Macsyma Equations. Master's thesis, University of California, Berkeley.

Foster, G. (1984a, April). DREAMS: Display REpresentation for Algebraic Manipulation Systems. Technical Report UCB/CSD-84-193, University of California, Berkeley.

Foster, G. (1984b). User Interface Considerations for Algebraic Manipulation Systems. Technical Report UCB/CSD-84-192, University of California, Berkeley.

Foster, K. R. (1993, November). 'Abstract' math made practical. *IEEE Spectrum*, 42–59.

Fournier, R., Kajler, N., Mourrain, B. (1993, September). IZIC: a Portable Language-Driven Tool for Mathematical Surfaces Visualization. In A. Miola (Ed.), *Proc. of DISCO'93*, Gmunden, Austria, pp. 341–353. LNCS 722, Springer-Verlag.

Frame Technology, Corp. (1989). *FrameMaker Reference Manual, Version 2.0* (a computer program). San Jose, CA.

Fuchssteiner, B., et al. (1993). *MuPAD Multi Processing Data Tool*. Birkhäuser. To appear.

Gardin, F., Campbell, J. A. (1981, August). Tracing Occurrences of Patterns in Symbolic Computation. In *Proc. 1981 Symposium on Symbolic and Algebraic Manipulation*, pp. 233–238. ACM Press.

Gardin, F., Campbell, J. A. (1983). A Knowledge-based Approach to User-friendliness in Symbolic Computing. In *Computer Algebra—Proc. EuroCal '83*, Lecture Notes in Computer Science 162, pp. 267–274. Springer-Verlag.

Genesereth, M. R. (1977a). An Automated Consultant for MACSYMA. In *Proc. 1977 MACSYMA Users' Conference*, pp. 309–314. NASA CP-2012.

Genesereth, M. R. (1977b). The Difficulties of Using MACSYMA and the function of User Aids. In *Proc. 1977 MACSYMA Users' Conference*, pp. 291–307. NASA CP-2012.

Genesereth, M. R. (1979, June). The Use of Semantics in a Tablet-Based Program for Selecting Parts of Mathematical Expressions. In V. E. Lewis (Ed.), *1979 MACSYMA Users' Conference*, pp. 328–335.

Gonzalez-Vega, L. and Recio, T., editors (1994, March). The PoSSo NEWSLETTER. Available electronically from `posso.dm.unipi.it`.

Graci, C., Narayan, J., Odendahl, R. (1989). Bunny Numerics: A Number Theory Microworld. In E. Kaltofen & S. M. Watt (Eds.), *Computers and Mathematics*, pp. 228–239. Springer-Verlag.

Gray, S., Kajler, N., Wang, P. S. (1994, July). MP: A Protocol for Efficient Exchange of Mathematical Expressions. In *ACM Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, Oxford, GB, pp. 330–335. ACM Press.

Hart, D. (1994). Function Browsers and help Stacks. *The Mathematica Journal* **4**(1), 26–27.

Hearn, A. C. (1984, April). *Reduce User's Manual, Version 3.0.* P.O. Box 2138, Santa-Monica CA 90406: The RAND Corp.

Heeman, F. C. (1990, September). Incremental parsing of expressions. *The Journal of Systems and Software* **13**(1), 55–70.

Hicks, T. G. (1985). *Standard Handbook of Engineering Calculations.* McGraw-Hill.

Hoffman, C. W., Zippel, R. E. (1979a). An Interactive Display Editor for MACSYMA. In *Proc. 1979 MACSYMA Users' Conference*, pp. 344.

Hoffman, C. W., Zippel, R. E. (1979b). MACSYMA Display-Oriented Expression Editor Preliminary Description and Command Summary. Handed out at 1979 MACSYMA Users' Conference.

Jenks, R. D. (1984, July). A Primer: 11 Keys to New Scratchpad. In J. P. Fitch (Ed.), *EUROSAM'84*, Cambridge, England, pp. 123–147. LNCS 174, Springer-Verlag.

Jenks, R. D., Sutor, R. (1992). *AXIOM, the Scientific Computation System.* Springer-Verlag.

Johnson, S. C. (1978). *UNIX Programmer's Manual* (Second ed.), Volume 2b, Chapter Yacc: Yet Another Compiler-Compiler. Bell Laboratories.

Kaiser, G. E., Kant, E. (1985). Incremental Parsing without a Parser. *Journal of Systems and Software* **5**, 121–144.

Kajler, N. (1990, April). Building Graphic User Interfaces for Computer Algebra Systems. In A. Miola (Ed.), *Proc. of DISCO'90*, Capri, Italy, pp. 235–244. LNCS 429, Springer-Verlag.

Kajler, N. (1992a, April). Building a Computer Algebra Environment by Composition of Collaborative Tools. In J. P. Fitch (Ed.), *Proc. of DISCO'92*, Volume 721 of *LNCS*, Bath, GB, pp. 85–94. Springer-Verlag.

Kajler, N. (1992b, July). CAS/PI: a Portable and Extensible Interface for Computer Algebra Systems. In P. S. Wang (Ed.), *Proc. of ISSAC'92*, Berkeley, USA, pp. 376–386. ACM Press.

Kajler, N. (1993a, Mars). Environnement graphique distribué pour le Calcul Formel. Thèse, Université de Nice-Sophia Antipolis, Ecole Doctorale SPI.

Kajler, N. (1993b, November). User Interfaces for Symbolic Computation: a Case Study. In *Proc. of the ACM Symposium on User Interface Software and Technology*, Atlanta, Georgia, pp. 1–10. ACM Press.

Kajler, N. (1994, December). IZIC 1.0.: an Overview for the User. *Computer Algebra Nederland (CAN) Newsletter* **13**, 17–25.

Kajler, N. (Ed.) (1995). *Human Interaction for Symbolic Computation.* Texts and Monographs in Symbolic Computation. Springer-Verlag. To appear.

Kajler, N., Monagan, M. (Eds.) (1996). *User Interfaces and Protocols.* Special issue of the Journal of Symbolic ComputationXS. To appear.

Karian, Z. A. (Ed.) (1992). *Symbolic Computation in Undergraduate Mathematics Education*, Volume 24 of *MAA Notes.* Washington, D.C.: The Mathematical Association of America.

Katz, A. (1987, May). Issues in Defining an Equations Representation Standard. *ACM SIGSAM Bulletin*, 19–24.

Kernighan, B., Cherry, L. (1978). *UNIX Programmer's Manual* (Second ed.), Volume 2b, Chapter Typesetting Mathematics—User's Guide. Bell Laboratories.

Knuth, D. E. (1984). *The TEXbook.* Addison-Wesley.

Krausz, F. (1988, July). A Better User Interface for Symbolics Lisp Machine MACSYMA. *MACSYMA Newsletter* **5**(3), 3–5.

Krausz, F. (1989, July). A Better MACSYMA User Interface. *MACSYMA Newsletter* **6**(3), 10–13.

Lamagna, E. A., Hayden, M. B., Johnson, C. W. (1993, September). The Design of a user Interface to a Computer Algebra System for Introductory Calculus. In *French–Norwegian Symposium on Computer–Aided Mathematics*, Gran, Norway.

Lamport, L. (1993, December). How to Write a Long Formula. Technical Report 119, Digital SRC.

Leler, W., Soiffer, N. (1985, August). An Interactive Graphical Interface for Reduce. *ACM SIGSAM Bulletin* **19**(3), 17–23.

Leong, B. (1986, July). Iris: Design of an User Interface Program for Symbolic Algebra. In *Proc. 1986 Symposium on Symbolic and Algebraic Computation*, pp. 1–6. ACM Press.

Martin, W. A. (1967, January). *Symbolic Mathematical Laboratory.* Ph. D. thesis, M.I.T., MAC-TR-36.

Martin, W. A. (1971, March). Computer Input/Output of Mathematical Expressions. In *Proc. of the Second Symposium on Symbolic and Algebraic Manipulation*, pp. 78–89. ACM Press.

Marzinkewitsch, R. (1991, July). Operating Computer Algebra Systems by Handprinted Input. In S. M. Watt (Ed.), *Proc. of ISSAC'91*, Bonn, Germany, pp. 411–413. ACM Press.

MATHLAB Group (1977). *Macsyma Reference Manual, version nine.* M.I.T., Cambridge, MA.

MathSoft, Inc. (1989, April). *MathStation, Version 1.0* (a computer program). 201 Broadway, Cambridge, MA, 02139, USA.

MathSoft, Inc. (1993). *MathCAD 4.0 user's Guide* (a computer program). 201 Broadway, Cambridge, MA, 02139, USA.

McCarthy, E., Holland, C., Lehman, J. (1987, August). *The Publisher User Manual* (MIC 3.2.7 ed.). Ann Arbor, MI: ArborText Inc.

Microsfot Corporation (1993). *Word User's Guide, version 6.0* (a computer program).

Microsoft Press (1994). *OLE 2 Programmer's Reference, Volumes 1 and 2.* Microsoft Press. ISBN 1-55615-628-6 and 1-55615-629-6.

Minsky, M. L. (1963, November). MATHSCOPE: Part I—A Proposal for a Mathematical Manipulation-Display System. Technical Report MAC-M-118, Artificial Intelligence Project, Project MAC, MIT.

Moses, J. (1971, March). Algebraic Simplification: A Guide for the Perplexed. In *Proc. of the Second Symposium on Symbolic and Algebraic Manipulation*, pp. 282–304. ACM Press. Also available in Communications of the ACM, 14(8), 527–537, August 1971 (Section on Lexicographic Ordering not in revised version).

Nakayama, Y. (1989, May). Mathematical Formula Editor for CAI. In *Proc. CHI '89*, pp. 387–392. ACM Press.

Notkin, D. (1985). The GANDALF Project. *Journal of Systems and Software* **5**, 91–105.

OpenMath Group (1994, March). OpenMath Specification. Available from `http://www.rrz.uni-koeln-.de/themen/Computeralgebra/OpenMath/`.

Orth, D. L. (1990). Linearizing Handwritten Mathematical Notation. In *Submitted to SIGPLAN '90 Conference on Programming Language Design and Implementation.* ACM Press.

Ossanna, J. F. (1978). *UNIX Programmer's Manual* (Second ed.), Volume 2b, Chapter Nroff/troff User's Manual. Bell Laboratories.

Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit.* Addison-Wesley.
Pasanen, H. (1992). Highly interactive computer algebra. Master's thesis, Helsinki University of Technology.
Pintur, D. A. (1994). MathEdge: The application Development Toolkit for Maple. *MapleTech* **1**(2), 31–38.
Porter, G. J., Hill, D. R. (1994). *Linear Algebra in Mathcad (prelimivary edition).*
Quint, V. (1983). An Interactive System for Mathematical Text Processing. *Technology and Science of Informatics* **2**(3), 169–179.
Quint, V. (1984, October). Interactive Editing of Mathematics. In *Proc. First International Conference on Text Processing Systems*, pp. 55–68. Boole Press Limited.
Quint, V. (1989). *Structured documents*, Chapter Systems for the representation of structured documents, pp. 39–73. The Cambridge Series on Electronic Publishing. Cambridge: Cambridge University Press.
Revesz, G. E., Lynch, K. T. (1991, July). A Context-Free Characterization of Ordinary Mathematical Notation Encoded in TEX. Technical Report RC 16615, IBM Research Division, T.J. Watson Research Center.
Rich, A., Stoutmeyer, D. (1979). Capabilities of the MUMATH-78 Computer Algebra System for the INTEL-8080 Microprocessor. In *Symbolic and Algebraic Computation, Proc. EUROSAM '79*, Lecture Notes in Computer Science 72, pp. 241–248. Springer-Verlag.
Rimey, K. (1992, April). Template-based Formula Editing in Kaava. In J. P. Fitch (Ed.), *Proc. of DISCO'92*, Volume 721 of *LNCS*, Bath, GB, pp. 1–12. Springer-Verlag.
Robb, T. (1992). *InterCall.* PO Box 343, Subiaco 6008, Perth WA, Australia: Analytica.
Sammet, J. E. (1969). *Programming languages: History and fundamentals.* Series in Automatic Computation. Prentice-Hall.
Scheftic, C. (1993). *Mathematical Computation with Maple V: Ideas and Applications* (Tom Lee ed.)., Chapter Interactive Mathematics texts: ideas for developers, pp. 51–63. Boston: Birkhauser.
Schelter, W. F. (1987). Sample $\mathrm{I_N F_{\oplus} R}$ Display. Unpublished Manuscript, Dept. of Mathematics, Univ. of Texas, Austin.
Skiena, S. (1990). *Implementing Discrete Mathematics.* Addison-Wesley.
Smith, C., Soiffer, N. (1986, July). MathScribe: A User Interface for Computer Algebra Systems. In *Proc. 1986 Symposium on Symbolic and Algebraic Computation*, pp. 7–12. ACM Press.
Smith, D. C., Irby, C., Kimball, R., Verplank, B., Harslem, E. (1982). Designing the Star user interface. *Byte* **4**, 242–282.
Soft Warehouse, Inc. (1991). *The DERIVE User Manual: A Mathematical Assistant for Your Personal Computer, Version 2* (a computer program). 3615 Hardling Av. Honolulu, HI 96816.
Soiffer, N. M. (1991, April). *The Design of a User Interface for Computer Algebra Systems.* Ph. D. thesis, University of California, Berkeley, CA. Available as report UCB/CSD/ 91/626
Stallman, R. (1979). Emacs, the Extensible, Customizable Self-Documenting Display Editor. Technical Report 519, Artificial Intelligence Lab, MIT.
Steen, L. A. (Ed.) (1988, October). *Calculus for a New Century: A pump not a filter*, Volume 8 of *MAA Notes.* Washington, D.C.: The Mathematical Association of America.
Suppes, P., Ager, T. A., Berg, P., Chuaqui, R., Graham, W., Maas, R. E., Takahashi, S. (1987, April). Applications of Computer Technology to Pre-College Calculus: First Annual Report. Technical Report 310, Institute for Mathematical Studies in the Social Sciences, Stanford.
Swinehart, D. C., Zellweger, P. T., Beach, R. J., Hagmann, R. B. (1986, October). A Structural View of the Cedar Programming Environment. *ACM Trans. On Programming Languages and Systems* **8**(4), 419–490.
Talbot, T. (1992). *EquationBuilder User Manual.* 100 Charles St., Ste D, Boston, MA: Digital Tool Works.
Teitelbaum, T., Reps, T. (1981, September). The Cornell Program Synthesizer: A Syntax-Directed Programming Environment. *Communications of the ACM* **24**(9).
Tektronix, Inc. (1988). *MathScribe User's Manual Version 1.0.* 164 pages.
Tyhurst, T. R. (1993). Mathematical Output Presentation in User Interfaces for Computer Algebra Systems. Master's thesis, University of Waterloo, Ontario. Available as Technical Report CS-93-05.
Upson, C., Faulhaber, T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., van Dam, A. (1989, July). The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Application* **9**(4), 30–42.
van Egmond, S., Heeman, F. C., van Vliet, J. C. (1989). INFORM: an Interactive Syntax-Directed Formulæ-Editor. *Journal of Systems and Software* **9**, 169–182.
Venable, D. (1985). *MacEqn* (a Macintosh computer program). Rochester, New York: Software for Recognition Technologies.
Vielhaber, H. (1992, July). A Graphical Usre Interface for a Computer Algebra System. Master's thesis, RISC-Linz, Johannes Kepler University. Available as RISC-Linz Report Series No. 92-48.
von Sydow, B. (1992). The design of the Euromath system. *Euromath Bulletin* **1**(1), 39–48.

Wang, P. S. (1990, April). A System Independent Graphing Package for Mathematical Functions. In A. Miola (Ed.), *DISCO'90*, Capri, Italy, pp. 145–154. LNCS 429, Springer-Verlag.

Wells, M. B. (1972). A Review of Two-Dimensional Programming Languages. In *Proc. of a Symposium on Two-Dimensional Man-Machine Communication*, pp. 1–10. ACM Press. Also available in SIGPLAN NOTICES, 7(10), October 1972.

Wells, M. B., Morris, J. B. (Eds.) (1972, October). *Proc. of a Symposium on Two-Dimensional Man-Machine Communication*, Volume 7 of *ACM SIGPLAN Notices*.

Wolfram, S. (1988). *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley.

Wolfram Research, Inc. (1988). MathTalk Reference Guide. Mathematica Technical Report.

Wolfram Research, Inc. (1993a). MathLink Reference Guide. Mathematica Technical Report.

Wolfram Research, Inc. (1993b). *User's Guide For the X Front End*.

Young, D. A., Wang, P. S. (1987). GI/S: A Graphical User Interface for Symbolic Computation Systems. *Journal of Symbolic Computation* **4**, 365–380.