

# A Review of Mathematica

Richard J. Fateman  
Computer Science Division  
Electrical Engineering and Computer Sciences Dept.  
University of California  
Berkeley, California

November 10, 1993

## 1 Introduction

The Mathematica<sup>1</sup> computer program is a general system for doing mathematical computation [54]. It includes a command language, a programming language, and a calculation environment that is oriented toward symbolic as well as numeric mathematics.

It is claimed to be revolutionary (“Mathematica will revolutionize the teaching and learning of math...” *Steven Jobs*. The New York Times says it “... fundamentally alters the mechanics of mathematics.”) *Fortune* says “... it will do, instantaneously, virtually all of applied mathematics ... ” [51]. See also [27].

Hype aside, the program is without question interesting to mathematicians, computer scientists, and engineers because of its combination and integration of a number of technologies which have arisen in initially separate contexts – numerical and symbolic mathematics, graphics, and modern user interfaces. Its most dramatic component, the exploitation of the PostScript(tm) language for plotting, contributed to its natural fit into the package of programs initially released for the NeXT workstation. Attention to standard graphics-window interfaces for Macintosh and X-window-based systems has added to its gloss.

Not all commentary on Mathematica has been uncritical. For example, reviews in *Science* [18] and *Notices of the AMS* [26] compare Mathematica’s features, reliability and efficiency to similar programs for algebraic and/or numerical interactive manipulation. Electronic-mail messages on various semi-public bulletin boards (in particular **netnews: sci.math.symbolic**) have discussed features and bugs of Mathematica as well as similar programs. There is also an active mailing list specifically for Mathematica users (**mathgroup@yoda.ncsa.uiuc.edu**). Discussions in such forums provide opportunities for valuable exchanges, but especially as “subscribers” become more numerous, such continuing unedited message streams overwhelm the large picture. Therefore there appears to be value in a more widely available and more detailed commentary on Mathematica, specifically from the perspective of its context and contribution to technology.

We believe there are three areas of technology to which Mathematica makes a potential contribution:

- the field of symbolic and algebraic computation,
- programming language and human-computer interface design, and
- the organization of mathematical information.

## 2 The Origin of this Review

This review has been composed (and, perhaps, decomposed) over several years, and is largely a combination of numerous rather orthogonal commentaries. We have despaired of melding them into a completely continuous whole. Furthermore, in collating these comments, we have, in the interests of brevity, occasionally assumed the reader has a more-than-casual familiarity with Mathematica. Although we hope we have kept such material to a minimum,

---

<sup>1</sup> Mathematica is a trademark of Wolfram Research Inc.

we must caution persons only slightly familiar with Mathematica — some sections requiring Mathematica-exotica may have to be skipped.

Finally, we expect that for some readers, Mathematica is beyond reproach. From such a perspective we undoubtedly will seem excessively critical of the system. While each reader is entitled to judge whether we are “fair,” we should note up-front that we have seen at least three reasons for people to champion a computer system (Mathematica or any other) that they have been using.

- There is a “Svengali” effect, whereby the designer of a system (or other highly skilled practitioner) puts a system through its paces before an audience, demonstrating the program’s apparently unbounded capabilities. Indeed, by knowing exactly what the limits are, and (consciously or not) skillfully avoiding them – or bluffing through them – such a performance can be very effective. The designers of all systems exhibit this to some extent<sup>2</sup>. Members of the audience are naturally impressed by such demonstrations.
- There is an “oops, what did I do wrong?” effect. After achieving enough skill to interact with an initially unfamiliar, yet clearly powerful, system, you may adopt a position that the system’s capabilities extend smoothly in all directions: in the case of an algebra system, users feel that all of mathematics is within easy reach. They then blame any shortcomings (even outright bugs) on their own limited understanding of the program. For these reasons, casual users do not, in general, realize that rather than reaching to the mathematical horizon, the capabilities of computer algebra systems (Mathematica as well as its major competitors) exhibit dangerous pits rather close to the “origin.”
- “Cognitive Dissonance” – once you’ve gone to the effort to learn one system, you have a natural tendency to view it as better than alternatives. This holds for automobiles, spouses, and computer algebra systems.

### 3 Prior Art in Mathematica

The idea of using computers for symbolic rather than numerical (arithmetical) computation, actually predates the electro-mechanical computer. Ada, Countess of Lovelace, suggested such usage in 1844 [31]. It took over a century for the first actual symbolic computation to be cited in the literature. (see van Hulzen [52] or Barton and Fitch [3] for a survey.)

The more recent tradition of the field of Symbolic and Algebraic Manipulation (SAM) by computer has had a small but loyal following at least since the early 1960’s. Members of the Association for Computing Machinery (ACM) joined together to form a Special Interest Group (SIGSAM) in 1965.

Chronologically, Mathematica probably should be considered as about “third-generation” among algebra systems, placing it among the (more-or-less) contemporary general-purpose systems such as Derive, and Maple, and the second-generation hold-overs, Macsyma, Reduce 3, Scratchpad II, and a few others. Wolfram’s previous effort, SMP, might be perhaps at generation 2.5. These systems were built upon research results plus practical experience in using first and second-generation systems of the late 1960’s, including ALTRAN, CAMAL, FORMAC, Mathlab, Scratchpad I, Symbolic Mathematical Laboratory, SAC-1, SIN, and others. An early comparison of some of these systems is still worth reading for background [3]. The first generation systems of the early-to-mid 1960’s included ALPAK, FORMAC, Formula Algol, PM, SAINT, SNOBOL and LISP. There are numerous other “special purpose” systems such as Schoonschip, Trigman, Cayley, and others, described in the references.

Although there are ample historical precedents for Mathematica’s symbolic facilities, and a clear intellectual debt, there is virtually no acknowledgment of prior software or algorithms in Wolfram’s reference [54]. If you want to know what the “Risch algorithm” is (mentioned on page 528) or how factoring is done, you won’t find any information on it.

This may be viewed as one of the costs of “technology transfer” – other software packages sometimes neglect to provide such background information; they, on the other hand, do not ordinarily claim such encyclopedic coverage or novelty.

Although space limitations prevent us from providing details and full references, recent developments in user interfaces (the Macintosh environment, other window systems, systems such as Soiffer’s work at Tektronix [46] and Arnon’s work at Xerox [2], Bonadio’s Macintosh program “Theorist” [5]), Avitzur’s Milo [4], operating systems (interprocess communication), display technology (PostScript in particular), and programming language ideas (object-oriented programming, pattern matching, functional programming) have also influenced Mathematica.

---

<sup>2</sup>but few have engaged in whistle-stop promotional tours of their systems.

The appearance of vastly improved low-cost computing hardware has made many types of resource-intensive computing practical. Since large amounts of memory are often needed for symbolic computation, Mathematica seems to have arrived at an auspicious juncture: sufficient numbers of Macintosh II computers with adequate memory (as well as adequate speed) became available in a timely manner. Other platforms including workstations and Intel 80386 computers were soon to follow.

Strong claims have been made for Mathematica, and we examine a number of them explicitly in this review. In some cases Mathematica performs better than other systems mentioned above, in others it does about as well, and in some cases it falls short. In the ways it falls short, it is sometimes the immodesty of the claims that is most troublesome<sup>3</sup> It is hard to anticipate being totally satisfied with a program claiming to “do mathematics”.

## 4 Examination of the Objectives

Stephen Wolfram, in his user-documentation for the system [54] specifies several objectives for Mathematica which we have summarized or paraphrased below. In our review we will try to address each of the objectives, and provide general commentary<sup>4</sup>.

### Objectives of Mathematica

- To provide a system for doing interactive symbolic mathematical calculations (interactive Mathematica); [section 5]
- To provide a repository for mathematical exposition and education (Notebooks); [section 6]
- To provide a programming language which unifies ideas from procedural programming, functional programming, rule-based programming, object-oriented programming and constraint-based programming; [sections 7 and 8]
- To provide facilities for exact and arbitrary-precision numerical computation; [sections 9, 10]
- To provide a repository for information on the simplification and manipulation of mathematical functions, polyhedral objects, etc. (Libraries); [section 11]
- To provide high-quality plotting from algebraic or discrete computations in a format that can be further manipulated (PostScript); [section 12]
- To provide built-in functionality (data types) for algebraic manipulation of formulas comprising polynomials, rational functions, the usual functions of elementary calculus, advanced functions of physics (“special functions”), functions of number theory, combinatorics, as well as composite data structures (lists, matrices, literal functions and arguments, etc.) and debugging;
- To provide built-in operations for algebraic manipulation such as symbolic differentiation, integration (in closed form), summation, approximation in Taylor series, etc. (as well as operations that can be extended by programming).

We also discuss some general issues about the use of canonical forms in algebraic manipulation, and the potential for Mathematica developers to repair errors. Not unexpectedly, we conclude with some conclusions.

---

<sup>3</sup>Even though some of the claims are not made by WRI, they contribute to the advertising.

<sup>4</sup>When it is appropriate, we will provide some comparison to other systems; however, the claims of Mathematica are made in relation to mathematics, and rarely with reference to other computer programs. It seems appropriate to try to review the system in that light. Consequently, we will not have tables of timings or feature-by-feature comparisons.

## 5 The Interactive System

Mathematica is intended to be used primarily as an interactive program to support the day-to-day computational needs of a mathematician or scientist. As such, most implementations fit into a traditional model typical of the last 25 years of time-sharing, or the last 10 years of standard data entry into personal computers. The user types a line or more, and after some computation, a display is produced. Given the possibilities that have developed recently, it is somewhat surprising that Mathematica hasn't progressed much beyond line-at-a-time input; for an example of what more could be done, see Milo [4], Theorist [5], or MathScribe [46]. Each of these systems allow some use of pointing devices for selection of mathematical expressions. (By contrast, experimental input systems based on tablets and character recognition have just recently resurfaced as "palm-top" computer systems. The intuitive attractiveness of entering mathematics for raw input – rather than selecting already-displayed material – has never been successfully exploited in past experimental systems. The combination of typing and pointing seems to work better.) In editing of commands, Mathematica allows selections only of linear text-strings. In its parser as well as its display of equations (limited to fixed-width character-grid typewriter-font "2-D" expressions) Mathematica seems more like Macsyma (circa 1968) than a system taking advantage of bit-mapped workstations.

Because of the interactive nature of most use of Mathematica, the intuitiveness of the system and the user-visible programming language is very important. We discuss the two parts of it in separate sections on the display (in the next section) and a more extensive subsequent discussion of the programming language.

An advertised novelty in Mathematica is its separation of a front end "interactive" component from a back-end. In most cases both parts of the program run side-by-side in the same computer, but in principle, they could be running on distinct machines. (The notion of a Macintosh front-end attached to a supercomputer back-end seems to have potential primarily for "hype" – supercomputers generally do not run symbolic mathematics programs particularly well, and in my experience running both front- and back- end parts of Mathematica on a fast remote computer works well). Mathematica was not the first system to try this separation since there were prior experiments with the Maple system. Even so, the separation has still-unrealized potential along with difficulties<sup>5</sup>.

Mathematica is potentially appropriate for use as an interactive front-end to other programs written in C or other languages. Through a "foreign function" interface, it is possible to link to other systems. It appears to be non-trivial to get this to work, however.

## 6 Notebooks and the Display

Mathematica runs on a variety of machines, and the quality of the front-end varies substantially from the universal but workable ASCII-terminal simulator through "X-window" supported systems to the highly-tuned versions on the Apple Macintosh and NeXT lines of computers.

On the Mac, the model for interaction is of typing into a text file which, when viewed at a higher level, is really an outline processor with the option of suppressing details of display at lower levels. The sections in the Macintosh Notebook can be straight text (not mathematical), Mathematica programs and commands, and Graphics sections. The graphics sections can be re-displayed as PostScript and edited. The Notebooks can be exchanged. This provides a technique for reproducing results and building up a library – if the Notebooks are properly constructed so as to not conflict in the user's space of objects. Although this is undoubtedly a useful approach, in some respects it is not as advanced for modeling of mathematical problem solving as some other programs in attempting to harness the relative inflexibility and speed of the computer with the extremely flexible, but slow and inaccurate pencil and paper type of calculation.

It would be nice to see  $\text{\TeX}$  or similar quality typesetting in the text part of the notebook: it is natural to hope that the descriptive material could use some of the features so well-supported on the Macintosh computer. It is possible to import descriptive material from other programs, including pictures or formulas, but the linkage is rather roundabout. Perhaps in the future it will be possible to run a  $\text{\TeXForm}$  version of an equation through  $\text{\TeX}$  and put it in the notebook in a simple fashion.

Programs that go considerably further in modeling pencil and paper computation include Milo and Theorist [4], [5].

These programs offer facilities missing in Mathematica: they allow the "text" parts of the mathematics to be data objects in a system where algebraic expressions can be selected, manipulated, linked to other expressions, etc.

---

<sup>5</sup>For example, the band-width for interaction can be tricky – can the back-end read the location of the mouse? When two programs compete for scarce resources, unexpected problems can crop up on some systems. The Macintosh implementation was initially subject to memory problems.

Since there are various authors set on development of educational material using Notebooks, we will presumably be able to see more examples of their use in the near future. Notebooks seems more supportive of presentation of information than interaction, and this may be just fine. One colleague finds the Notebooks to be the best new feature of Mathematica compared to other symbolic mathematics systems. On the other hand, another serious Mathematica user indicated to me that he found the front-end to be a hindrance.

Some of the difficulties may be traced to the split of computation between the “front end” and “back end” facilities. Mathematica uses its own protocol, and (for example) X-window based facilities use an X-protocol. The latter facility may be more highly developed and integrated into the engineering workstation environment of the future than the one-at-a-time hand-crafted interactions of the Mathematica developers. Certainly the Macintosh version is currently better than the X-window version, but will that always be the case?

## 7 Programming Language

### 7.1 Overview

Even before the circa-1960 development of languages such as Fortran and Algol-60, there was a belief among computer programmers that there should be a universal language for communicating with computers. Although this model has not always been explicitly expressed, and is out of favor at the moment, it still has some advocates. Some scientists felt that, with Fortran, “programmers would be obsolete” because every scientist would do whatever programming was necessary. Of course this hasn’t happened, and the experiments continue. The challenge, phrased as a programming language issue, is how to create the right syntax and semantics so that all the problems of applications programs vanish. The approaches, through the years, have developed into three streams:

- The all-inclusive language (like PL/I, Ada, Common Lisp)
- The bare-bones but extensible language (Algol-68, C)
- The specific application-oriented language.

Mathematica has taken most constructions from general languages and most extension techniques from extensible ones, and put them into one system. It is actually fairly comforting to an experienced programmer – most of the familiar tools, as well as some others – are there. It also provides meat for puzzlers to explore computationally equivalent programs written using different programming paradigms which are orders of magnitude different in resource consumption. The fact that there are such variations may be disconcerting to mathematicians who feel they have solved a problem by displaying *any* construction of a proof. Now they may have to search for an efficient construction – one that depends on knowledge of algorithms and data structures that may not be explained anywhere.

Mathematica uses all the symbols of the ASCII character set, and quite a few multi-character symbols. Whether this leads to natural problem solving or “runic programs” is debatable, but the possibility of “write-only” programs (which cannot be comprehended after they are written) arises. Mathematica’s designers did not choose the more conventional wisdom that it may be advisable to leave some characters “for the users” – extensible syntax in (for example), Macsyma and Common Lisp means that for applications, unanticipated constructions can be added later.

Mathematica builds a complete syntactic box, for good or ill.

From the mathematician’s point of view, J. R. Kudera [34] comments “... computer mathematics *languages* are ghastly to use” and that “Problems that lend themselves to this kind of computation [user-written programs] simply do not occur often enough to allow users to develop proficiency.” Thus intuitiveness is important: to the extent that gaining proficiency requires extensive deviation from common mathematical notation and semantics, the system design is poor. A novel programming language or notation runs the risk of tripping up the user who either writes new code or must read others’ code. When there is a substantial deviation from *both* programming and mathematics a system may be positively surprising, if not hazardous.

Nevertheless, programming seems inevitable since the system constructors simply cannot anticipate all needs. How can the system support programming, then?<sup>6</sup>

---

<sup>6</sup>Incidentally, computer scientists who have abandoned the search for a “universal solvent” for all programming problems have largely moved on to research in “environments”. This may be even harder, but is closer to what is needed.

## 7.2 Object-Oriented Programming

For the specialist, we merely point out that Mathematica's version of this popular notion does not support hierarchies and inheritance. This omission considerably weakens the faithfulness to the notion. It is nicely integrated with the pattern matcher, however.

### 7.2.1 Contexts

It is appropriate to provide information-hiding capabilities in a large system, and Mathematica uses its notion of Contexts for this purpose. It appears to resemble the "package" system in Common Lisp, and uses the delimiters **BeginPackage** and **EndPackage**. In Mathematica it is possible to delimit sections of code by **Begin** and **End** brackets, identifying a context in which public names (those exported to external or Global contexts) and private names (those local to this context) are separated. Unfortunately, this is less effective than one might wish, because entering and exiting a Context (by setting the `$ContextPath`) does not have the effect one might expect.

For example, one might wish to assert the rule (one which is often used as an example, and appears on the back cover of the reference book [54]), that logs of products should be re-written as sums of logs.

```
Log[x_*y_] := Log[x] + Log[y]
```

But just saying this in the Global context is dangerous<sup>7</sup>. In particular, in version 1.1, we found that with this rule in place, the **Integrate** command no longer worked on some examples<sup>8</sup> (`Integrate[LegendreQ[1,z],z]`). So one might wish to contain it in a context, for example:

```
Begin["logsimp'"]
Log[x_*y_] := Log[x] + Log[y]
End[]
```

However, this rule is placed on the Global **Log** symbol, rather than the `logsimp'Log` symbol, and the system does not distinguish between a rule that rewrites `Log[x_*y_]` and one which re-writes `Log[logsimp'x_*logsimp'y_]`. Thus this packaging does not limit the effect of the **Log** rule, and one must presumably explicitly delete and re-assert such rules when needed (or explicitly apply them when appropriate). Merely asserting them once for all time leads to generally unforeseeable consequences including possibly breaking the **Integrate** command. This appears to severely limit the utility of rule-based programming as a technique for adding general information to the system: Programmers must not attempt to use the same function symbols as the system.

A subtle point, perhaps not intended to be noticed by the casual fan of Mathematica, is that the rule on the back cover of [54] defines simplification for `log` not **Log**, thereby not interfering with built-in rules. This trick of using lower-case names does not work very well if one wishes to alter, by rules, any built-in functions, or functions like **Factorial** or **Plus**, which do not have lower-case equivalents in their usual representation as `!` and `+` respectively.

It is possible to group rules and apply them together, as illustrated by programs in the on-line library, as well as in Maeder's text [35]. Such grouping of rules as in the trig-simplification routines eliminates "cross-talk" by limiting scope. Unfortunately, such tricks weaken the possible synergy of rules. If the idea behind rules is to have them take effect when appropriate, without specific attention by the programmer, the necessity for grouping vitiates the concept. One experienced Mathematica hand advised me not to modify any built-in functions. This simple piece of advice carries with it implicitly the idea that if you don't like what Mathematica does with the **Log** function, you are free to program anything and everything you want about the `log` function. But then you'll have to change **Integrate** which returns answers in terms of **Log** to (say) `integrate` which returns answers in terms of `log`. If you choose to differentiate the result, you have a choice of changing `log` to **Log** temporarily, or propagating your changes throughout your program: defining rules for the differentiation and numerical evaluation (etc.) of `log` – in effect writing a "shadow" Mathematica. This is made rather difficult because the system's internal functioning is proprietary, and it is difficult to see how much functionality must be recreated.

Two other shortcomings in Contexts are worth noting: Mentioning a name before reading in the package defining it shields the name in the package from the global environment, effectively disabling the package. Debugging, never easy in Mathematica, becomes even harder when packages and contexts are involved. Also, printing out a program defined in another context (see, for example, the data associated with the **Bessel** functions) involves the repeated display of fully-qualified and rather lengthy context names.

<sup>7</sup> It provokes a warning that is overcome by "Unprotecting" the symbol **Log**.

<sup>8</sup> This particular problem was fixed in version 1.2. Whether similar problems have been guarded against, generally, is doubtful.

## 7.2.2 Spaces mean multiplication, and other oddities

This is perhaps a minor point, but annoying in its own way. In the Mathematica programming language, one can use spaces or even adjacency to signal multiplication. This idea, used by Wolfram in his earlier SMP system, is initially appealing – that one can simply write  $2x$  or even  $2x$  instead of  $2*x$ . It looks like the traditional mathematical convention. But is it? Consider that it implies that  $\sin x$  or even  $\sin(x)$  is a product, equal to  $x * \sin$ . The Scratchpad II system, following another mathematical convention, interprets  $\sin x$ ,  $\sin(x)$  and  $\sin.x$  as function applications. Mathematica requires the syntactic construction  $\sin[x]$  or  $\sin@x$  or  $x//\sin$  for function application, and just to make sure you use the built-in function, you must capitalize the first letter:  $\text{Sin}[x]$ . This departure from conventional notation is more of a concern for the casual user, but then, most potential users have not had much experience typing conventional mathematics. On such reasoning, requiring square-brackets may not be objectionable.

What else goes wrong, though?

A few things. For example,  $a++ * ++b$ , which to those familiar with the C programming language appears to be the computation of  $a*(b+1)$  leaving  $a$  set to  $a+1$ , as well as  $b$  to  $b+1$ , cannot be written in Mathematica as  $a++ ++b$ , since *that* space does not stand for multiplication. Rather it stands for nothing: Mathematica (versions 1.1 and 1.2) parses this expression as the necessarily meaningless  $(a++)++ * b$ . Even the authors of the program were confused on this, since echoing back the first expression (by typing `Hold[a++ * ++b]` yields the non-equivalent `Hold[a++ ++b]`). (The inconsistency of the display with the internal form is presumably easy to fix – in this review we eschew reporting mere bugs in Mathematica because there are so many and reporting them would be distracting – but it is a mis-feature that the parser and the display can even use inconsistent precedences. In a well-designed system the two related subsystems would use the same precedence data, stored in one place).

The precedence of the space as multiplication is not adhered to. (As another example,  $3! ++a$  results in an error: `Factorial is write protected`).

As a matter of clarity, I suspect the physicist who is used to writing  $1/2\pi$  may be lured into writing  $1/ 2\text{Pi}$  which is actually the rather different  $\pi/2$ .

My conclusion is that insisting on the use of the asterisk is preferable, because without it, too many problems come up. One reader suggests as an alternative, inserting parentheses when there is any doubt. Unfortunately, many people have no doubt when they should, and therefore leave off parentheses.

## 7.2.3 Other Syntactic oddities

The version 1.2 valid input form  $(a,b)$  is not documented. It is not a `List`, but `Sequence[a,b]`. Sequences are most naturally produced by pattern matches involving the double-blank. The `Head` function does not work on Sequences, and so it may be difficult to detect such a data type. A Sequence has the remarkable property that  $f[1,2,Sequence[a,b]]$  means  $f[1,2,a,b]$ . One consequence is that  $f@(a,b)$  is mapped to the same form as  $f[a,b]$ , but the perhaps easily mis-typed  $f(a,b)$  is somewhat unexpectedly mapped into  $a*b*f$ . (Explanation:  $f(a,b) = f*(a,b) = \text{Times}[f,Sequence[a,b]] = \text{Times}[f,a,b]$ ). If you type  $f(0,x)$  you get 0. A simple fix for this problem is to be a little less clever and not allow the alternative input-syntax of  $(a,b)$  for `Sequence[a,b]`. It appears that in version 2.0, WRI has taken my advice. You still should beware that  $f(0)$  regardless of the definition of  $f$  is 0. As another example, if you want to redefine factorial for certain values, you might think to do `Unprotect[Factorial]; big!=bigfact`, but that won't work. It turns out that you need the space: `big! =bigfact` because there is a not-equals operator (`!=`).

## 7.3 Procedures, functions, patterns

I like the approach used in Mathematica, except for the errors in semantics. In a nutshell, the approach is to

- abandon the notion that the language should be uniformly intuitive to mathematicians, but provide superficial correspondence to conventions from elementary algebra, trigonometry, and parts of calculus.
- incorporate control structures from all of C, Lisp, APL and functional programming.

Difficulties with details crop up, and we discuss some of them, especially with respect to patterns, later.

## 7.4 Rules and Patterns

Basically, any algebraic system that tries to implement mathematics by transformations on (mostly) uninterpreted trees, is going to fall into pits. Consider the plausible rule of  $x_1 - x_2 := 0$ . This might be considered universally true, regardless of the pattern which  $x$  matches. Yet it is not true where each (apparently identical) syntactic expression can really be different. For instance, it is probably an error to simplify `Random[Integer,10] - Random[Integer,10]` to 0.

Similarly, two occurrences of  $O[x]^2$  are not semantically identical, because each implies a (perhaps different) asymptotic constant. In particular,  $O[x]^2 - O[x]^2 = O[x]^2$ . (In fact, the use of the equality for that expression is an unfortunate convention; a more formalistic approach would use a notation perhaps reminiscent of set inclusion. This is perhaps more obvious when one observes that the statements  $f(x) = O(x)$  and  $g(x) = O(x)$  do not imply  $f(x) = g(x)$ .)

The problem here is fairly deep. How could one modify the rule that  $x_1 - x_2 := 0$ ? Perhaps by saying that  $x$  must satisfy some predicate? What should that predicate be? Mathematica supports examination of the **Head** of the expression tree that is denoted by  $x$ , and this will sometimes work. If  $x$  is an integer, we might agree the rule applies. But if  $x$  denotes (say) an expression headed by **Plus**, one must examine in principle, all components to see if any of them are dangerous. Mathematica has no handle on this problem, and the kind of handle that is necessary is probably based on global information regarding the type of an expression. This problem is eliminated by systems which assign types to expressions, such as Scratchpad II.

How are rules ordered? We are told that the most explicit rules are used in preference to the most general, yet it is clear that the ordering of explicitness needs to compare domains, something that Mathematica does not do very well. Roman Maeder ([35] p. 59) is more up-front about this. "...Mathematica cannot always find out which of the rules is more special than the other and it might fail to reorder them accordingly."

Patterns are intimately related to the definition of rules. The 1, 2, and 3-blank "match variables" with optional attached predicates is compact and relatively easy to read. The handling of defaults appears more general than other computer algebra systems; the handling of commutative operators is probably no messier than necessary. One can hope that the implementation is no costlier than necessary. The integration of the pattern matcher into the language extension facilities is neat. The idea of so-called "up-rules" to avoid clogging common operators (especially **Plus** and **Times**) with rules, is a clever heuristic. This allows, for example, rules that pertain to the addition or multiplication of special functions to reside with the special functions, not with the common operators. This means that in principle, the simplification of sums is not slowed down unnecessarily by having to look at inapplicable rules, even if they are nominally rules affecting sums. These problems have been addressed, for the most part less effectively, by numerous earlier programs. (A sample would include [9], [14], [23], [25], [29], [36], [37])

Yet all is not well. Consider, for example, the transformations to contract certain expressions involving factorials: Simplify  $n(n-1)!$  to  $n!$ . A rather economical and readable version of this (given that one must first understand patterns) is

```
n_ * (n_ - 1)! := n!
```

Placing this rule on **Factorial** rather than **Times** is a good idea:

```
Factorial/: n_ * (n_ - 1)! := n!
```

since this would only affect the speed of simplification of products involving **Factorial**. A version that is more generally applicable, and works for example on  $(h-2)(h-3)!$  is

```
Factorial/: n_ * m_! := n! /; n==m+1
```

Indeed, we discovered a rule equivalent to this latter piece of code in Mathematica's combinatorial simplification library, along with some slightly more ambitious rules:

```
Factorial/:
```

```
(n_)!/(m_)! := Product[i, {i, m+1, n}] /; n - m > 0 && IntegerQ[n-m]
```

```
Factorial/:
```

```
(n_)!/(m_)! := 1/Product[i, {i, n+1, m}] /; m - n > 0 && IntegerQ[m-n]
```

```
Factorial/:
```

```
(k_)! k1_ := (k1)! /; k1 - k == 1 (*equivalent to our rule *)
```

Consider now the problem of reducing  $n/(n!)$  to  $1/(n-1)!$ . It would seem that a rule

```
Factorial /: n_/(n_!) := 1/(n-1)!
```

would do the trick. Unfortunately, one gets the error message:

```

TagSetDelayed::notag: Tag Factorial not found in -----.
                    (n_)!

```

This means that the rule cannot be placed on `Factorial` because it is not among the top two levels of Heads in the left-hand-side. Indeed, only `Times` and `Power` qualify, and neither one of those is an attractive choice, being too common. (The expression looks like `Times[n_,Power[Factorial[n_],-1]]`). The point of this illustration is that the up-rule idea only delays by one level the inevitable exponential growth of rule-sets to implement complex simplifications.

Consider reducing to zero the expression

$$m!(m+1)^2 - (m+1)!^2.$$

The rules do not work, and one presumably has to try for a new rule involving powers, perhaps of the form

```
n_^e_ * m_!^f_ := n^f1[e,f]*m!^f2[e,f] /; pred(e,f,n,m)
```

where the details of the predicate to be applied, as well as the functions `f1` and `f2`, are, for the moment, unimportant. (Properly formulated, this rule covers one of the earlier rules.) Again what is significant here is that the “uprule” technique won’t work two levels down: this rule must either be placed on `Times`, or on `Power`, and neither option is attractive. Thus, while the Mathematica pattern matcher is handy for patching the simplification rules, it is probably unwise to expect this technique to provide simple and efficient implementation of all new code. It is a false illusion that has been pursued rather strenuously by two decades of symbolic manipulation programs that mathematical knowledge can be imparted primarily by rule-transformations on trees. The more fundamental approach of transforming expressions into canonical forms when possible has been shown to be far more effective than the *ad hoc* melding of rules, regardless of heuristics for speeding up pattern matching. (For factorials this is probably best done by a calculus of difference and shift operators.)

## 8 Major Semantic Problems

### 8.1 Canonicity

Wolfram argues ([54], pages 212-213) that it is satisfactory to not have a program that reduces expressions to canonical form, because no single program can do this for the full range of expressions that one can use in Mathematica. This is deceptive, because for very significant types of calculations there are adequate, and some could argue, required canonical forms. By not providing ANY canonical forms, even for ratios of polynomials, the user is left somewhat at loose ends. Particular problems involving complicated expressions and their negatives inside radicals are repeatedly cited as difficulties in public “bug reports” in the network newsgroup `sci.math.symbolic`. We discuss this further in section 15.1.

### 8.2 Dummy arguments are patterns

Mathematica combines function definitions and pattern matching with an interesting technique. A simple program definition `f[x_]:=...` looks like a pattern-match for the expression `f[...]`. This simplifies the definition of a function over distinct structural cases (or type of arguments) by allowing more elaborate “dummy argument” specifications. Thus one can define `f` on a special case expression where `x_` would match `Sin[...]` as `f[Sin[y_]]:=...`. One can define `f` for integer arguments by `f[x_Integer]:=...`

Among algebraic manipulation systems perhaps Scratchpad I is a precursor, although other programming languages including ML, Prolog, and SNOBOL have similar notions. Another place the merging of the concept of matching and parameter passage appears is in the destructuring of lambda-list arguments to `defmacro` in Lisp [48]. Unfortunately, the analogy, or the implementation of the analogy, leaves a bit to be desired. In Macsyma, the program which uses the dummy variable as a local parameter

```
f(x):=[x,x:x^2,x:x^2]$
f(2);
```

produces the list `[2, 4, 16]`. The Mathematica version, because it uses “macro-expansion” style parameters, has a different interpretation:

```
f[x_]:= {x,x=x^2,x=x^2}
f[2]
```

produces an error message concerning the setting of “Raw object 2” (namely  $2 = 4$ ). Is this a feature? This may be related to the design error explained in the next section, or might be independent.

### 8.3 Binding of variables

Here is a simple program:

```
g[x_]:=Block[{a=x},a=1+a]
```

This is a program much simplified from what one might write in the course of computing an elaborate function, and it seems to be very straightforward. The result of `g[s]` is `1+s`. But why then is `g[a]` not `a+1`? It is `251+Hold[1+a]`<sup>9</sup>, with a warning message `Recursion depth of 256 Exceeded`. This is one result of a Mathematica “feature” which can have drastic destructive effects:<sup>10</sup> Any time you compose (presumably a more serious) program with local variables, you may have a conflict with a name that occurs within the actual parameters. In the Mathematica manual, section 2.5.10, Advanced Topic: Function Arguments and Local Variables, Wolfram admits, “Sometimes it can be very confusing to have the name of a local variable conflict with a value you give for a function argument. The best way to solve this problem is somehow to have the names of the local variables that appear in your functions be such that they will never appear in the values of function arguments.” That is, you could try to avoid this conflict by programming, for example:

```
g[x_]:=Block[{afunnyvar},afunnyvar=1+x]
```

But this can rapidly get tiresome, and it is not really fully effective anyway.<sup>11</sup>

This is a serious semantic matter for programmers using Mathematica. Section 2.7 of the manual muddies the issue, as does Roman Maeder [35]. Maeder suggests avoiding the accidental “capture” of variable bindings by using the construction

```
f::usage = "...
Begin["Private'"]
f[x_] := Block[{a},a=x+1]
End[]
```

The user’s variable `a`, or to be more verbose, `Global'a` is then never the same as `Global'Private'a` that is used inside the body of `f`. However, typing `f['Private'a]` still causes an error, so the problem is not really solved. You can elaborate on the Context names to make it less likely to conflict (you then have to worry about two programmers accidentally choosing the same Context name), but you cannot really eliminate the problem as is done by, for example, Pascal. If you make the mistake of trying to look at a program in some Context, on-line, you will be deluged by the repetition of Context qualifications on every name. This is quite painful.

As another example, the definition `h[j_] := Sum[h[k],{k,0,j-1}]` has problems with the local variable `k`. Mathematically speaking it is reasonable that `h[0]` returns 0, and it would seem that `h[1]` should return `Sum[h[k],k,0,0]` or just `h[0]`, which we know is 0. However Mathematica 1.2 writes

```
General::itervar: In iterator {k, 0, 0 - 1}, variable k already has a value
```

followed by a 230-line message related to infinite recursion. The semantics of local variables in this particular expression seem to have been fixed in later versions of Mathematica, but even in version 2.0, `a` had better not have a value if you write `Limit[f[a],a->b]`.

As a sidelight, might one really wish to write a program `g[x_,y_] := (x=y)` which when invoked as `g[r,3]` actually sets the value of `r` to 3? Programming by side-effects on parameters is nearly universally condemned, and is certainly unnecessary.

---

<sup>9</sup>in version 1.2, I got `254 + Hold[Hold[1+a]]`.

<sup>10</sup>To quote from Maeder ([35] p. 8) “The more subtle ones [problems] become only apparent in the context of a longer session with Mathematica and are then normally very hard to find.” Actually, it appears that his proposed solution is a work-around for a bug. Furthermore, he only defers the problem by using packages.

<sup>11</sup>An effective and also tiresome way would be to contrive names that no-one else could *possibly* use, even in the *future*. Oddly enough, the Lisp language, via the `GENSYM` function provides this just this facility. The Mathematica `Unique` function only provides a name not used in the past. An effective and totally transparent solution to this is provided in any language with appropriate evaluation and variable-scoping rules. Pascal, for one.

## 8.4 Evaluation

Evaluation of variables is a complex problem in algebraic manipulation systems ([41], [19]). We've already noted that Mathematica's model, based on "infinite evaluation" probably contributes to a basic difficulty in using local variables. But there are other funny issues. It appears to be impossible to "Quote" a name. This construction is highly useful in Lisp, and presumably in any one-level evaluation symbolic language. Presumably the Mathematica team deliberately chose infinite evaluation, but the justification is not apparent. Large programming efforts usually require some level of "information hiding" to prevent separately-developed modules from colliding. Infinite evaluation interferes with this. M. Monagan has pointed out (from experience with the Maple system) that some operations which the user might expect to be constant time are instead linear time with such a model.

Once you have executed `x=3`, in a Mathematica session, you cannot refer to the name `x` in any context except explicitly as `Hold[x]`. If you use the function `Release` on that object, you get 3. Given a chain of assignments `a=b; b=c`, any mention of `a` results in a 2-level evaluation, returning `c`. Oddly, removing `b` (via `Remove[b]`) changes the meaning of `a` to `Removed[b]` in version 1.2, but leaves it alone in version 2.0. How does one make `b` stand for the literal `b`?

This infinite evaluation scheme makes it imperative that you know the full dependencies of any value you might use. Given, for example, the earlier chain of assignments, would you realize that any change to `c` would also change `a`? There appears to be no way to determine the dependency.) Since this area of the Mathematica system (the issue of `Hold`, `Release`, and related constructions) seems to be in flux, perhaps the authors will eventually find a better choice of semantics.

## 9 Numerical calculation

Mathematica associates a Precision and an Accuracy with each number. A variable has these attributes only to the extent that they are associated with its value.

Mathematica defines Accuracy as the number of decimal (!) places to the right of the decimal point.

There are many puzzles resulting from the design. I am not particularly expecting these to change, although a few blunders undoubtedly will be fixed. Making the model more useful would require an admission of the incorrectness of the one being promoted, and after much fruitless discussion on this point, it appears that WRI and I simply do not agree.

Accuracy is associated with the location of the decimal point, although (in version 1.1) all "small" numbers appear to be Accurate to double precision (16 places or so). (Let us assume this does not affect us, by adding 17 extra places to the left to fool the system). In Mathematica 3.01 meters is more Accurate but less Precise than 301.0 centimeters.<sup>12</sup>

In version 1.1, the numbers `x=123456789012345678901.0` and `y=1234567890123456789012.0` are each of Accuracy 2 (Why 2? Perhaps in conversion to binary, some "Accuracy" was added), but `100*x` has Accuracy 5. Maybe this is just a bug<sup>13</sup>. The difference `10 x - y` is -2, Accuracy 5<sup>14</sup>.

Mathematica does arbitrary-precision arithmetic, presumably when necessary. But then why should certain exact integer arguments cause problems for the `Sin` routine? For example, in version 1.1, `Sin[31415926535897932.]` which is about -0.37521, instead is computed as 0, Accuracy 16, with a message `sin: TLOSS error`. And then `N[Sin[Floor[10^50*N[Pi,50]]],21]` gives a warning about overflow and returns a value (in version 1.1) of about  $1.309 \cdot 10^{78400}$ , and in later versions 0 with accuracy -59. For all real numbers, the sine is known to be between -1 and 1. To not know the sine to within  $\pm 10^{59}$  is very pessimistic. (There are several packages for computing arbitrary precision transcendental functions reasonably correctly in the open literature [7], [55].)

Much of the manipulation of the big-O asymptotic notation used in `Series` was problematical in version 1.1. For example `Sin[0[x]^4]` never returned. Probably the issue of combining significantly distinct not-entirely algebraic datatypes is difficult to handle in the Mathematica scheme. Even in the latest version, which says `Sin[0[x]^4] = 0[x^4]` and `Sin[Pi/2+0[x]^4] = 1+0[x^4]`, erroneously believes that `Cos[0[x]^4]=1`.

Consider the function `f[h_]:=N[N[Pi,h]-N[Pi,2*h],3*h]` which one might think would give the difference between an h-digit value for  $\pi$  and a 2h-digit value, computed to 3h digits. Computing `f[50]` we find the value 0,

<sup>12</sup>In discussions with WRI employees, they defend this by reiterating "Accuracy is defined as the number of significant digits to the right of the decimal place." and that "Your paradox results from associating the everyday meaning of accurate with the well-defined concept of Accuracy." This is reminiscent of Lewis Carroll: "When I use a word," Humpty Dumpty said in rather a scornful tone, "it means just what I choose it to mean - neither more nor less" [*Through the Looking Glass*, 1872]

<sup>13</sup>Apparently they each have Accuracy 1 in version 1.2, so the earlier result must be a bug.

<sup>14</sup>Also changed, to 0, in version 1.2.

(Accuracy 58, Precision 58). For `f[120]` the value is  $6.10^{-126}$ , (Accuracy 125, Precision 1). Clearly only about  $h$ -digit computation is being done. Version 1.2 of Mathematica has built-in functions `SetAccuracy` and `SetPrecision` which probably do not compensate completely for doing it wrong in the first place<sup>15</sup>.

Before trying to explain what I think should be done, I'd like to quote some remarkable puzzles uncovered by David Jacobson of Hewlett Packard Laboratories. (quoted with permission)

Suppose that I double a floating point number then subtract off the original number. I ought to get the original number back, with maybe the least-significant-bit (LSB) being trashed. If I do this  $n$  times, the introduced error should be at most  $n$  times the weight of the LSB. Well, Mathematica's high precision floating point numbers don't work that way! In fact, the way they work is downright hazardous, as I found out.

```
In[1]:= a=1.1111111111111111;
In[2]:= Do[a=2a-a,{55}];
In[3]:= a
Out[3]= 0.
```

Wow, just 55 iterations got a zero, not a 1.111... with some error. Furthermore that zero is a black hole:

```
In[4]:= a+1
Out[4]= 0.
```

Everything works fine if we use a machine float instead of a big float:

```
In[5]:= b=1.111111111;
In[6]:= Do[b=2b-b,{1000}];
Out[7]= 1.11111
In[8]:= b-%5
Out[8]= 0.
```

The problem is that each iteration ratchets down the precision by one until it eventually gets to zero.

Although it is less serious, there are also constructs that ratchet up the precision, which, by the way, indicates that the precision calculation cannot be quite correct.

```
In[35]:= p=N[Pi,20]
Out[35]= 3.1415926535897932385
In[36]:= Do[p=(p+p)/2,{100}]
In[37]:= p
Out[37]= 3.1415926535897932384626445913472373472822072149898
In[39]:= Precision[p]
Out[39]= 50
In[40]:= N[Pi,50]
Out[40]= 3.1415926535897932384626433832795028841971693993751
```

You might think that you never use big floats, so you don't have to worry. Unfortunately, the function `Fit` sometimes outputs a function with coefficients that are big floats, even when only passed machine floats in the input.

---

<sup>15</sup>It seems that Mathematica should allow one to say `Accuracy[x]=20`. It allows it, and it looks like it works, but it doesn't really. Mathematica makes it easy to fake such surface appearances without appropriate adjustment to the operation of the system.

The ratcheting up of precision may be a performance problem, but the ratcheting down of precision is downright disastrous. I can suggest two approaches: either take care to avoid bigfloats altogether in calculations where each iteration uses the results of the previous iteration, or if you really need higher precision, figure out in advance what precision you want, then restore the precision of every variable at the end of each iteration with something like `x = SetPrecision[x,desiredPrecision]`

The immediate problem arises from the way Mathematica computes the precision/accuracy of the result of an addition: it seems to make the accuracy of the result be the accuracy of the least accurate addend.

There are at least 4 models we might seriously consider for floating-point arithmetic that can be used in the arbitrary-precision bigfloat world.

#### **Version 0.**

Consider 3.0 as a representation (presumably in a floating-point format in the computer) of the exact integer 3. That's it. No special hidden "precision" or "accuracy".

A consequence of this is one can express a bound on the error of a value by using two such numbers. For example, if  $x$  is in the interval  $[2.94, 3.01]$ , then  $x$  lies in the closed interval between the two exact numbers  $294/100$  and  $301/100$ . This notation need not be restricted to decimal representations, of course, (since Mathematica provides both exact rationals and other-radix forms, and probably uses binary internally, it is odd to see Accuracy defined in decimal.) An interval might be  $[1/3, 2/3]$ . Open intervals, not including end-points, are also objects that can be manipulated. If you want to express the result of a non-rational operation, you can express the answer as an interval. Another possibility is to consider that `N[Sin[N[x,p1]],p2]` is the result of the following sequence of operations: evaluate  $x$  to  $p1$  decimal digits (or the roughly equivalent number of bits). Treating this as an exact rational number, evaluate the sine function with sufficient accuracy to provide  $p2$  decimal digits of the answer. That too is an exact number, and it is related to the sine function in a way that can be analyzed. (Incidentally, the sine of an interval is *not* computed by "running the algorithm for computing the sine function in interval arithmetic". Such an automatic conversion could easily give you an interval larger than  $[-1, 1]$ . There is a substantial literature on interval arithmetic.)

#### **Version 1**

Consider 3.0 as a representation of any number greater than 2.05 and less than 3.05. That is, any number which rounds to 3.0 when printed in two decimal digits. The numbers which you might type in as 3.0 and 3.00 are different. The numbers might be printed out more-or-less in this fashion.

Perhaps an implementation of the physicist's notion of accuracy (a measurement is good to  $N$  decimal places and therefore has accuracy  $N$ ) is the source of this idea. Unfortunately, most computations don't really work this way. To continue with our sine example, regardless of how uncertain you know the (presumed real) argument to sine, you know the result is still between -1 and 1.

#### **Version 2**

Accuracy or Precision of the number 3.0 is merely a bound on the error introduced in each computation (including conversion to floating point) leading up to the present value. Thus a precision of 16 digits (53 bits) means the result is as accurate/precise as could have been expected if 53 bit mantissas had been used in all calculations leading to the number. This says nothing about how close the number 3.0 is to the real number that would have occurred if all arithmetic in an entire computation were exact; rather, it says that in each individual computation at most so much error is introduced by the arithmetic (and by forcing results back into variables of this much "precision"). This is not much different from the concept of "computing in single precision" or "double precision" except that there is a continuously variable precision.

#### **Version 3**

Accuracy or Precision of the number 3.0 is an upper bound on the relative difference between the number 3 actually represented and that which would have occurred if each computation were exact. Thus 3.0 with a precision of 4 means that, had all arithmetic been exact, the result would have been in the range 2.9997 and 3.0003.

Version 0 has been implemented in Macsyma's bigfloat package. It appears that Mathematica uses something like version 1. We performed a small experiment in Mathematica version 1.2. We set the "top-level" so that each number was printed as a triple: {value, Accuracy[value], Precision[value]}, and computed a few expressions:

```
In[1]:= p[x_]:= {x, Accuracy[x], Precision[x]};
In[2]:= a=10^50
```



```

Out[1]= 
$$\frac{1 - x \cos[t]}{1 + x^2 - 2 x \cos[t]}$$


```

```
In[2]:= Integrate[%,{t,-Pi,Pi}]
```

```
Out[2]= 0
```

Actually, the value of this integral depends on the value of the parameter  $x$ . For example, if  $x = 1/2$ , Mathematica 1.2 correctly computes the answer as  $2\pi$  rather than 0.

In a later version of Mathematica, the errors caused by too quickly simplifying the square-root of perfect-squares that are generated in this problem, are avoided by not simplifying roots. The answer is given as

$$\frac{\pi \left( 2 - 2x^2 + \sqrt{4(-1+x^2)^2} \right)}{\sqrt{4(-1+x^2)^2}}$$

This solution is perhaps too conservative in simplifying; the remedy is apparently to advise the user to apply **PowerExpand** to simplify and/or to commit errors. This has two benefits for the program authors: (a) The analysis necessary to see if the simplification can be done is now mostly unnecessary<sup>16</sup>; (b) Any errors in expanding powers are now committed explicitly by the user.

Neither of these benefits is of great use to the customer. Indeed, the most likely situation may be that the user is less well equipped than the system to determine the validity of the transformations in **PowerExpand**.

Incidentally, the integration answer is still arguably wrong for  $x = 1$ . Substituting 1 for  $x$  in the answer gives **Indeterminate** as a result of computing a  $1/0$  expression. Using the **Limit** program gives 0 even though the directional limits from above and below give different answers of 0 and  $2\pi$  at  $x = 1$ , and finally, if you compute the integral with  $x = 1$  from first principles the answer is  $\pi$ .

(Much more can be said along these and related lines. Prof. W. Kahan of my department has collected a bundle of errors committed by Mathematica, and to a somewhat lesser extent, most other algebra programs.)

## 11 Libraries

The packaging mechanism seems to be an unhappy one, perhaps no worse than that available in other languages such as Common Lisp. Common Lisp however seems to have a more effective technique for organization of modules in its Object System (CLOS). Interactive programs that intend to have access to large libraries of scientific programs include Matlab, Gauss, and numerous other PC or time-sharing packages. Systems that include both non-trivial symbolic mathematics and numerics include MathStation and SENAC. It is imperative that the routines be of the highest quality, and not just some “fill in the blanks” routines. Because Mathematica has arbitrary precision floating point numbers, the analysis of the correct routine is sometimes difficult. If the numerical model in Mathematica were cleaner, the programs might be safer. Alternatively, a model of fixed precision might be adopted for numerical routines, making it possible to utilize *en masse* the product of several decades of scientific computing analysis.

Apparently there is a situation in which certain programs must be loaded manually before the system has any information about them. If such programs can be loaded manually, why not by the system? Why is it that the system presupposes that the user knows how to load the appropriate packages needed to do an integral? Can the program know that a routine that is not loaded can or cannot do a computation?

As a framework for the integration of mathematical knowledge, Mathematica is rather disappointing. This is a comment on the interface issue rather than the storage and retrieval of the information. (As a separate issue, the incorporation of large tables of integrals using Mathematica’s pattern-matcher is problematical in terms of effectiveness and speed.)

### 11.1 Plotting

The adaptive plotting of curves in a plane is rather effective, choosing more points at areas of high curvative than other places, and rendering exceptionally distant points off-scale. Undoubtedly any plotting programs can be fooled

---

<sup>16</sup> Although  $\sqrt{3^2} = 3$ ,  $\sqrt{\pi^2}$  is unchanged.

by some functions, given the generality of the language. Defining functions by rules, logical statements or other discontinuous expressions provides enormous opportunity for generating difficult problems. Fractal style functions can be specified. Even ordinary expressions provide nasty opportunities. Consider the curve (taken from a problem set by W. Kahan) to stymie plotting functions,  $1 + x^2 + 0.0125 \log(|1 - 3(x - 1)|)$  for  $0 < x < 2$ . Mathematica does better than other programs we tried in finding the very narrow slot in an otherwise parabolic (diabolic?) function, that at  $x = 4/3$  dives to  $-\infty$ . The slot's width is on the order of  $10^{-4}$  units.

It appears that the same strategy for adaptive plotting is not in place for surfaces, where it would presumably be even more useful. Surface plots provides substantial aesthetic challenges. An uneven grid might provide misleading clues for perception of distance or curvature. Graphics provides a never-ending list of possible tricks.

The surface rendering algorithms seem to be highly effective, and probably have a broader level of interest in applications.

Although plotting features may be useful in the total environment, they are substantially orthogonal to the symbolic mathematics. Other (purely numerical) systems such as Matlab [38] have high-quality plotting routines. Using Post-Script as a device-independent intermediate form for plots is perhaps novel. Note that PostScript's ability to draw splines is not used; only straight lines are used in plotting.

A feature that is missing from Mathematica, (but, for example, present in Milo [4] and Theorist [5]) is interactive re-plotting, where one can specify a "rubber-band" rectangular region of interest and have that section blown up (or replotted) for more detail. Indeed providing this facility might be a challenge to Mathematica implementation, given the separation of front- and back-end processing in the design. The linkage, which might in the most general case require figuring out what kinds of modifications the user has inserted in a PostScript text, and its relationship to mouse-positions, may be daunting. I understand that a cursor-following program which identifies numerical values on a graph has been implemented; this is a first step along this path. On the other hand, the interface in Theorist is both easier to use (requiring less command information for the default case), and more interactive with menu-driven or mouse-directed changes such as zooming in, changing the viewpoint on 3-d graphs, or coloration<sup>17</sup>.

## 12 Datatypes and operations

### 12.1 Expressions

We are told that in Mathematica, every object is an expression which can be used anywhere in the system. This is a considerable overstatement which has probably led the implementors to not check for appropriate classes of data when needed. Sometimes expressions either make no sense, or have to be treated rather differently depending upon details which may be unexamined in general. For example, if we define a function  $f(x)$  with a discontinuity at 0 and take the limit of  $f(x)$  as  $x$  approaches 0, we see that (in version 1.1) Mathematica is unprepared for the task, and tries to do what is reasonable for analytic functions, but unreasonable here. That is, it computes a Taylor series.

```
Limit[If[x==0,1,0],x->0]
```

gives `If[0[x]^4+False*x+0[x]^3+Equal^(0,0),[0,0],1,0]`. This kind of error can be fixed on a case by case basis, but the notion that the `Limit` evaluation program should be constructed as a collection of manipulations of data-representations without mathematical semantics, is fundamentally inadequate. Having fixed the Limits of `If`, what will the next example bring? As a somewhat random example, but anticipating the next section, consider `Limit[0[x]^4,x->3]`. This is, in my opinion, undefined. Mathematica 1.1 gives 0. The semantics for big-O notation (see, for example, Graham *et al* [22]) can be dealt with formally, but it also notes how the use of the big-O notation is inconsistent with the usual meaning of "=".

Perhaps related to the general representation decision is the fact that Mathematica does not have a canonical simplification policy. The `Simplify` command in version 1.1 works at most for algebraic (not transcendental, exponential, or log) expressions. Even something as simple as  $-(-1+q)/(1+q)$  withstands all the simplification programs, and is not reduced to  $(1-q)/(1+q)$ . The suggested reduction requires a bit of programming: `ExpandNumerator [ExpandDenominator [Together[%]]]`. In some other systems, the appropriate use of data structures would produce simplified expressions as a consequence of the abstraction and representation in use. We suspect that an equivalent of Macsyma's `RATSIMP` command would help users of Mathematica. (An alternative

<sup>17</sup>Theorist is much less ambitious in algebraic computation, but has substantially better typesetting capabilities. It is also tied, at least in its current distribution, to Macintosh computers with 1 megabyte of memory or more.

explanation suggested by one reviewer is that in fact Mathematics IS imposing a canonical form, and that one has a minus sign out front. A relative of `RATSIMP` for Macsyma, namely a form with optional factored forms<sup>18</sup> might have just this form.)

## 12.2 Series

Although Mathematica uses a single format for expressions based on prefix trees, at least one sub-area is specialized, and uses the standard form to encode data more like a list of terms. These are expressions with the head `Series`. We've just said that we advocate the notion that special mathematical notions deserve special forms, but we're going to complain about how this one was integrated into the system. It is hazardous.

Given two Series that agree to a certain order, their difference should be equal to the next (omitted) term. Yet `Series[Sin[x],{x,0,2}] - Series[Tan[x],{x,0,2}]` comes out 0, rather than  $0(x)^3$ . When we attempted to correct this by putting a rule on `Subtract` errors in other parts of the system cropped up. `N[Series[...]]` fails. This bug was fixed in version 2.0, but its presence is indicative of the kinds of patches that will haunt the system unless there is a more formal approach to special data types.

## 12.3 Other Datatypes

If another type of data structure, such as Poisson series – similar to Fourier series – were to be introduced, how would the addition of two different series be handled? Or even the addition of two series in different variables or about different expansion points? These are touchy areas and cannot be dismissed with a simple appeal to mathematical conventions. The informality of defining combinations are only partially available in the type-directed pattern invocation. The difficulty is that the default operations on novel data-types will fall into the pit of Mathematica's notion of generic operations. Leaving the generic operations unaltered is risky; modifying the generic operations requires skill and may slow the system down substantially.

Is there a good solution? The correspondence between mathematical concepts and data abstractions, and then between abstractions and representations, has a rather substantial literature, especially among the Scratchpad implementors. Formal systems take this seriously, and there are successes to be observed. Multiple representations in Macsyma add to its power and complexity, although Mathematica seems to have gone only part-way toward integrating the creation of new data structures into the system. It could do better.

## 13 Debugging

While it is not our primary objective to complain about of particular implementations of Mathematica, when a system design itself may be interfering with the ability to debug programs, it rises to the level where it should be noted in a general review.

Dealing with the standard kernel, on several versions of the system on engineering workstations, when you hit an interrupt key, you are thrown into a break in which you cannot examine values, compute values, or do anything but print the name of the (usually internal) program that is being executed, or the sequence of programs (without arguments), being executed. One also has the choice sometimes of continuing, aborting the computation or exiting from Mathematica. Sometimes you do not have all these options. This is rather meager information. Whether this is in fact a design problem or an implementation that can be improved is not clear. (I found that I interrupted the system several times in the course of its execution one system program, `$$LuxuryYacht`. This was relatively uninformative, as debugging information goes.)

It is possible that the lack of garbage collection (Mathematica uses reference counts) makes it difficult to compute within a break-point. It is apparently possible to find non-interruptible loops in Mathematica, in which case no debugging is really possible, and a “crash with loss of data” is about the only way to halt a computation.

Finding and reporting bugs (but not necessarily correcting them) has been assisted substantially by the volunteer efforts of Mathematica enthusiasts with access to the computer networks that link many universities and high-tech companies in the US and abroad. Steven Christensen of the University of Illinois has been providing facilities for the maintenance of archives of program libraries, the running of a “mailing list” to serve for posting alleged bugs, calls for help, and other information. A comparable service for REDUCE users, based at the RAND Corporation is also available. Maple users have a similar forum. Macsyma used to have such a organized service but it appears

---

<sup>18</sup>first proposed and implemented in the ALTRAN system.

to have faded. This kind of service is, in my opinion, quite valuable and should help promote a growth in maturity. One defect in the Mathematica situation is that the bug corrections seem to be decoupled from the reporting, and therefore repeated demonstrations of the same bug can flood the mail. Another hazard (common to all such forums) is that not all advice is correct, and some of it is outright silly. Excluding the non-networked community is unfortunate, but for the moment, inevitable.

## 14 Abuse of Mathematics

While many of Mathematica's intellectual ancestors make logical hash of mathematical ambiguities or boundary values, probably no other system has documentation so bold as to assert that the system, rather than mathematical tradition should determine the meaning of (for example) multiple-valued inverses ([54] page 358) or that it is the human user of the system who has primary responsibility to check the input (and perhaps the output) of the system. ([54] page 425: "You have to be careful, however, when the integration region contains a singularity." ).

For example, mathematicians should agree that `ArcSin[Sin[x]]` is periodic, since real `ArcSin` is bounded between  $\pi/2$  and  $-\pi/2$ , and `Sin` is periodic. If you plot this function in Mathematica 1.1, you get a straight line,  $y = x$ . This might be confusing to students of algebra or calculus. At a conference at Colby College in 1987, S. Wolfram asserted this behavior is a feature (of version 1.1). Oddly, even plotting `ArcSin[N[Sin[x]]]` produces a straight line. (This feature has been removed from version 1.2, and the system now provides a sawtooth plot.)

As another example, you expect that a modern computer program that claims to compute limits should do so at least as well as some 20-year-old systems written on comparatively small machines (for example [28]). It is an invitation to disaster for a system to assert by default that `Limit[f[x], x->a] = f[a]` in general, and information to the contrary has to be asserted by the user. The view that an unknown function has an unknown limit seems a good deal safer.

While one can be philosophical about this and (to quote W. N. Venables, "Like cars and knives and most other useful things, symbolic manipulation systems in general, and Mathematica in particular, are inherently dangerous and not for the reckless.") it is certainly possible to include some safety measures.

## 15 Which weaknesses are easily fixed? Which are permanent?

We leave this section for the end. Certainly one can find errors of implementation, or inefficiencies, or situations in which programs in Mathematica were written to be efficient but insufficiently general. Improving Mathematica's algorithms may fix certain problems, and if past experience is any guide, occasionally will insert new ones.

In well-defined areas such as the division of arbitrary-precision integers, the factorization of polynomials over the integers, and so on, one can assume that any identified bugs in Mathematica will eventually be repaired. Thus, other than reporting and classifying known problems of this kind to some maintenance person, and perhaps devising work-arounds, the customer has little choice but to wait for the fix to arrive<sup>19</sup>.

The more difficult problems have to do with errors of design, particular errors of implementation (blunders?), or unjustified claims. Each of these areas is reviewed below.

### 15.1 Errors of Design

We have already mentioned each of these area previously.

- Scope of names: blocks, packages, rules

We have already mentioned problems with local programming variables, but Mathematica seems to have a problem (at least in versions 1.1, 1.2 ) dealing with quantified mathematical variables. Most mathematicians would agree that the use of the literal  $x$  in  $\lim_{x \rightarrow a} f(x)$  is locally bound, and that this expression is entirely equivalent to  $\lim_{z \rightarrow a} f(z)$ . Unfortunately, if you've typed `z=3` you will get a message `Limit::lim: Limit specification 3 ->0 is not of the form x->x0`. On the other hand, `z = f[x]` followed by `Limit[z, x->a]` provides a puzzle: is the  $x$  inside  $f[x]$  the same as the  $x$  inside the `Limit`?

Let us merely say there is a confusion inherent in Mathematica among local and global programming variables and mathematical indeterminates with the same printed name.

---

<sup>19</sup>Or use another program.

- Model of numerical calculation

It is possible to perform arithmetic on approximate quantities of different precision by using the wider precision, and padding the lower precision with zeros (of whatever radix is being used in the representation). It is also possible to truncate or round the wider precision to be cruder, and perform only the crude arithmetic. Mathematica chooses the latter. The former is more general.

- Stark data model / lack of canonical forms

There are no canonical forms except for Series, and that is a mixed blessing. In the realm of rational functions, it is a source of wrong answers: programs which depend on zero-equivalence being decided give wrong answers. Any routine that does divisions can fall prey to this problem.

M. Monagan points out that for some functions, e.g. `LinearSolve`, one can pass a `ZeroTest` to the function. It doesn't stop Mathematica from returning wrong answers, but merely "passes the bug" on to the user. Other competing systems at least provide normal forms (zero-equivalence testing) for rational functions over rational fractions.

It may be difficult and expensive for Mathematica to repair this error, since there are some 180,000 lines of C code in it (version 1.2) already. By contrast, Maple 4.3's kernel is 20,000 lines of C, with considerably greater capability in a higher-level language.

## 15.2 Errors of Implementation

- Difficulty of debugging
- Non-uniformity in names in spite of substantial evidence of the imposition of authority on naming conventions. There are still oddities (What would you use the term `doublyinfinite` for? Why is it `SetAccuracy[x,10]` and not `Accuracy[x]=10`? Because `SetAccuracy` returns a new number, and does not change the old one. Attributes are Global.)
- Reference counts? Their use is defended as though the absence of garbage collection makes storage allocation free or cheaper. No evidence is presented for why the technique used is superior, and we suspect that the use of reference counts contributes to the lack of debugging resources.
- Source code is not available, making certain behaviors even more mysterious. Maple is proprietary, but most of the source code is available.
- How many lines of C code will there be in version 2.0, and how can one expect it to be maintained? Will Mathematica outgrow all those 4 megabyte Macintoshes that it just barely fits on now (in 1990)? One person has suggested that storing old bit-maps is the culprit here, but there may be other reasons to grow, in the future.

## 15.3 Unjustified Claims

From the academic point of view, this last type of error has great potential for damage: there are many problems for which Mathematica appears to claim complete solutions, but is not even as good as other programs (for example, in symbolic integration, solution of equations, simplification). One might erroneously believe that if Mathematica cannot solve a problem, no other program (or human, perhaps) can do so. Users of Mathematica may prematurely abandon the technology because the first system they tried was insufficient. (This phenomenon was prevalent among early FORMAC users, who typically ran out of memory on the 32K word IBM7094 rather rapidly, and assumed that if this (at the time, quite powerful) IBM system couldn't solve their problem, it was not solvable).

There is a risk that the audience for symbolic mathematical computation will believe that "whatever Mathematica has done, has been 'done' as best as possible" and thereby believe that any shortfall is inherent in the technology.

On the other hand, the general awareness of symbolic computing has been vastly improved, and this may be more than compensation<sup>20</sup>.

---

<sup>20</sup>I am reminded of the story that the Xerox corporation described its use of a bit-map display, including the window techniques it pioneered, as "Macintosh-like" in deference to the public awareness of the widely advertised, but admittedly derivative system. Will the "SAM" systems of the past be referred to as Mathematica-like?

## 16 Conclusions

This review is hardly the final word on Mathematica, but merely a commentary on a program that has changed substantially during the writing of this commentary. (Perhaps in part as a reaction to earlier drafts.)

One reader (an employee of WRI) asked why I ever mentioned anything having to do with an “obsolete” version of the system. The answer is two-fold. Some people will continue to use the old versions; also looking at the flaws of the past gives some insight into the flaws of the future.

It is possible that, having raised the consciousness of the public to symbolic mathematics, the Mathematica program will then also evolve to satisfy all the various criticisms we’ve indicated. Alternatively, it may be supplanted by something correct with respect to these issues, newly written from the ground up. It is not clear that the technology exists today to follow either of these paths to completion.

The future will undoubtedly see more efforts in combining symbolic mathematics in a general scientific information and programming environment with applications for research and development, teaching, and even entertainment.

## 17 Acknowledgments

Opinions expressed in this paper are my own, and do not necessarily represent the views of government sponsors or others mentioned below. I thank Paul Abbott, Robert Campbell, Steven Christensen, Sam Dooley, David Jacobson, Dan Grayson, Velvel Kahan, Roman Maeder, Kevin McIsaac, Michael Monagan, Steven Omohundro, Malcolm Slaney, Neil Soiffer, William N. Venables, Stephen Wolfram, and others, for enlightening discussions.

A preliminary copy of this paper was provided to Wolfram Research Incorporated, and the resulting comments were used in refining this version.

The writing of this review has stretched out over several years, during which time this work has been supported in part by the following grants: National Science Foundation under grant number CCR-8812843 through the Center for Pure and Applied Mathematics, University of California at Berkeley; the Defense Advanced Research Projects Agency (DoD) ARPA order #4871, monitored by Space & Naval Warfare Systems Command under contract N00039-84-C-0089, through the Electronics Research Laboratory, University of California at Berkeley; the IBM Corporation; a matching grant from the State of California MICRO program.

## References

- [1] Harold Abelson and Gerald Jay Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA, 1985.
- [2] Dennis Arnon, Richard Beach, Kevin McIsaac, and Carl Waldspurger. *CaminoReal: An Interactive Mathematical Notebook*, Palo Alto Res Ctr, Xerox Corp, 1987. See also Report on the Works hop on Environments for Computational Mathematics ACM SIGGRAPH Conference 1987,
- [3] D. Barton and J. P. Fitch, A Review of Algebraic Manipulation Program and their Application, *Comput. J.* 15, 362-381 (1972). This is an extended abstract of “Applications of Algebraic Manipulative Programs in Physics,” in *Rep. on Prog. in Phys.* 35, no. 3 (1972) 235-314.
- [4] Ron Avitzur, “Milo” (a Macintosh computer program) Paracomp Inc. San Francisco, CA. 1988
- [5] Allan Bonadio, “Theorist” (a Macintosh computer program), Prescience Corp. 814 Castro St. San Francisco CA. 1990
- [6] B. Buchberger, G. E. Collins, R. Loos (eds), *Computer Algebra: Symbolic and Algebraic Computation*, Springer-Verlag, 1983.
- [7] R. P. Brent. “A Fortran Multiple-Precision Arithmetic Package,” *ACM Trans. on Math. Softw.* 4 no. 1, (March, 1978), 57-70.
- [8] Computer Algebra Group, “The Scratchpad II Computer Algebra System Interactive Environment Users Guide,” (Draft 1.1 July 19, 1988) Mathematical Sciences Dep’t, IBM Research Division, T. J. Watson Res. Ctr. Yorktown Hts, NY. (382 pages) see also R. D. Jenks and B. M. Trager, “A Primer: 11 Keys to New Scratchpad,” *Proc. Eurosam 84, Lecture Notes in Computer Science 174*, Springer-Verlag, 1984.

- [9] Gene Cooperman, "Semantic matcher for Macsyma," in B. W. Char (ed) *Proc. 1986 Symp. on Symbolic and Algeb. Comp.* ACM Univ. Waterloo, Ont. Canada, July 1986. (132-134)
- [10] Carl Engelman. "The Legacy of Mathlab '68" in *Proc. 2nd Symp. on Symb. and Algeb. Manipulation*, (ACM) 1971. (29-41).
- [11] Richard J. Fateman. "Macsyma's General Simplifier: Philosophy and operation," in V. E. Lewis (ed) *Proc. of the 1979 MACSYMA Users' Conf.* Washington, DC. June 20-22, 1979. (563-582)
- [12] Richard J. Fateman. "TeX Output from Macsyma-like Systems," *SIGSAM Bull.*, Aug., 1987.
- [13] Richard J. Fateman. "Advances and Trends in the Design of Algebraic Manipulation Systems," Proc. ISSAC-90 (ACM), invited paper, Tokyo, August, 1990, 60-67.
- [14] R. Fenichel. "An On-line System for Algebraic Manipulation," doctoral dissertation, Harvard University, July, 1966, also Report MAC-TR-35, Project MAC, M.I.T., available from the Clearinghouse, document AD-657-282.
- [15] John K. Foderaro. "Typesetting Macsyma Equations," MS Report, EECS Dep't., Univ. Calif., Berkeley, 1978.
- [16] John K. Foderaro. "The Design of a Language for Algebraic Computation Systems," Ph.D. dissertation, EECS Dep't., Univ. Calif., Berkeley, 1983.
- [17] Gregg Foster. "DREAMS: Display REpresentation for Algebraic Manipulation Systems," Rpt. UCB/CSD 84/193, Computer Science Div. Univ. of Calif, Berkeley April 1984.
- [18] Kenneth R. Foster and Haim H. Bau. "Symbolic Manipulation Programs for the Personal Computer," (Software review) *Science* 243, Feb 3, 1989, 679-684.
- [19] Jeffrey P. Golden. "The Evaluation of Atomic Variables in Macsyma," in C. M. Andersen (ed) *Proc. 1977 Macsyma Users' Conf.* Univ. of Calif, Berkeley, July 1977. (109-122)
- [20] Jeffrey P. Golden. "An Operator Algebra for Macsyma," in B. W. Char (ed) *Proc. 1986 Symp. on Symbolic and Algeb. Comp.* ACM Univ. Waterloo, Ont. Canada, July 1986. (244-246)
- [21] Gaston H. Gonnet. "An Implementation of Operators for Symbolic Algebra Systems," in B. W. Char (ed) *Proc. 1986 Symp. on Symbolic and Algeb. Comp.* ACM Univ. Waterloo, Ont. Canada, July 1986. (239-243)
- [22] R. L. Graham, D. E. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley Publ. Co. 1989.
- [23] J. M. Greif. "The SMP Pattern Matcher," in B. F. Caviness (ed), *Proc. Eurocal '85, vol. 2*, Lecture Notes in Computer Science 204, Springer-Verlag, 1985, 303-314.
- [24] A. C. Hearn. *Reduce 3 User's Manual*, The RAND Corp. P.O. Box 2138, Santa Monica CA 90406. Small system versions available from Northwest Computer Algorithms, P.O. Box 1747, Novato, CA 94948.
- [25] A. C. Hearn. "A new REDUCE model for algebraic simplification," *Proc. 1976 ACM Symposium on Symbolic and Algebraic Computation*, August, 1976, (46-50).
- [26] Eugene A. Herman. Review of Mathematica, (also, discussion by J Barwise, J. J. Uhl, Jr., Paul Zorn *Notices of the AMS* 35, no. 9 (Nov, 1988), 1334-1349.)
- [27] Alan Hoenig "Mathematica, a program for various work stations and personal computers," (Review) *Math. Intell.* 12 no. 2 (1990) 69-74.
- [28] R. Itturiaga. Contributions to Mechanical Mathematics, Ph.D. dissertation, Comptr. Sci. Dep't., Carnegie-Mellon Univ., Pittsburgh, Pa., 1967
- [29] R. D. Jenks. "A pattern compiler," *Proc. 1976 ACM Symposium on Symbolic and Algebraic Computation*, August, 1976, (60-65).
- [30] R. D. Jenks. "SCRATCHPAD/360: Reflections on a Language Design," *ACM SIGSAM Bulletin* 13, no. 1, Feb., 1979, (16-26).

- [31] D. E. Knuth, *The Art of Computer Programming, vol 1. Fundamental Algorithms*, Addison-Wesley Publ. Co., 1969.
- [32] D. E. Knuth. *The T<sub>E</sub>X book*, Addison-Wesley, Reading, MA, 1984.
- [33] Knut Korsvold. "On-Line Algebraic Simplify Program," Stanford A.I. Project Memo 37, Nov. 1965, 30 p.
- [34] J. Robert Kudera, "Physics Made Easy," letter to editor, *Fortune* May 25, 1988.
- [35] Roman Maeder, *Programming in Mathematica*, Addison Wesley, 1989
- [36] Mathlab Group. *Macysma Reference Manual*, Lab. for Comp. Sci, MIT, Jan, 1983 (2 volumes: version 10), available also from the National Energy Software Center (NESC), Argonne, IL. Similar manuals are available from Symbolics, Inc., for example, version 11 (Symbolics, Inc.) Oct. 1985.
- [37] K. McIsaac. "Pattern Matching Algebraic Identities," *SIGSAM Bulletin*, 19, no. 2, (May, 1985).
- [38] Mathworks Inc. "Matlab" (a computer program). S. Natick, MA.
- [39] Joel Moses. "Algebraic simplification, a guide for the perplexed," *Comm. A.C.M.* 14, no. 8, Aug., 1971, (527-538).
- [40] Joel Moses. "Macysma: the Fifth Year," in *Proc. Eurosam 74*, Stockholm, Sweden, *ACM SIGSAM Bull.* 8 No. 3, August, 1974 (105-110)
- [41] Joel Moses. "The Variety of Variables in Mathematical Expressions," in C. M. Andersen (ed) *Proc. 1977 Macysma Users' Conf.* Univ. of Calif, Berkeley, July 1977.
- [42] Richard Pavelle and Paul S. Wang. "Macysma from F to G," *J. Symbolic. Comp.* 1, no. 1, p. 69-100.
- [43] Carl Ponder, "Augmenting expensive functions in Macysma with lookup tables," Chapter 9 in *Evaluation of "Performance Enhancements" in Algebraic Manipulation Systems*, Ph.D. diss. Univ. Calif. at Berkeley, Dept. of EECS, also UCB/CSD 88/453 p. 85-100.
- [44] Vaughn R. Pratt, "Top Down Operator Precedence," in *ACM Symposium on Principles of Prog. Lang.*, Boston, MA October, 1973. See also, a detailed memo (1977) "CGOL - An Algebraic Notation for MACLISP Users" distributed with the CGOL source code.
- [45] Harlan Seymour. "Conform, a conformal mapping system," MS Project U.C. Berkeley, EECS Dept. 1985. Also see the paper of the same title in B. W. Char (ed) *Proc. 1986 Symp. on Symbolic and Algeb. Comp.* ACM Univ. Waterloo, Ont. Canada, July 1986. (163-168)
- [46] C.J. Smith and N. Soiffer. "MathScribe: A User Interface for Computer Algebra Systems," in B. W. Char (ed) *Proc. 1986 Symp. on Symbolic and Algeb. Comp.* ACM Univ. Waterloo, Ont. Canada, July 1986. (16-23)
- [47] L. Spirkovska. "MUFIE - Macysma's User-Friendly Interactive Executive," MS. Project, EECS Dept, UC Berkeley, July, 1986.
- [48] Guy L. Steele Jr., *Common LISP the Language*, Digital Press, 1984.
- [49] Symbolic Computation Group, "Maple", Computer Science Department, Univ. of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (Vendor for Macintosh Computer: Brooks/Cole Publishing Co., 5611 Forest Lodge Rd, Pacific Grove, CA 93950)
- [50] R. G. Tobey, R. J. Bobrow, and S. N. Zilles. "Automatic Simplification in Formac," *Proc. AFIPS 1965 Fall Joint Comput. Conf.*, (1965) (37-52).
- [51] G. A. Taubes, "Physics Whiz goes into Biz," *Fortune*, April 11, 1988. 90-93.
- [52] J. A. van Hulzen, J. Calmet, *Computer Algebra Systems*, [6] 221-224.
- [53] Walter K. Vogel. *Mathematica 1.1. Biotechnology Software* (Mary Ann Liebert Inc Publ. NY) July-August 1989 2-7

- [54] Stephen Wolfram, *Mathematica – A System for Doing Mathematics by Computer*, Addison-Wesley Publ. Co., Redwood City, CA., 1988.
- [55] W. T. Wyatt, Jr., D. W. Lozier, and D. J. Orser. “A Portable Extended Precision Arithmetic Package,” *ACM Trans. on Math. Softw.* 2, no. 3, Sept., 1976, (209-229).