Review in 2009 of Richard Fateman's "Review of *Mathematica*" (1992)

Thomas Colignatus, August 25 2009

http://www.dataweb.nl/~cool

Summary

Richard Fateman, one of the authors of Macsyma / Maxima and in more than one respect one of the founding fathers of computer algebra and doing mathematics on the computer, gave a review in 1992 of *Mathematica* 2. Around 2002 he stated "I think a careful reading shows that the broad comments are still valid." In 2009 it may be useful to check on *Mathematica* 7. The 1992 review is copied here section by section with the Foxit Snapshot and each section is pasted into *Mathematica* and discussed so that we can see interactively what has been changed and what is the same, and what are general issues in computer algebra. Since Fateman's review takes 33 pages, we limit this review of his review to the first 12 pages. This already gives a wealth of insight. Conclusions are: (1) that Fateman provides a highly valuable review of *Mathematica* 2 and general points in computer algebra, still relevant for *Mathematica* 7, (2) that some points may be a bit overdone but most points are on target, (3) that *Mathematica* 7 has taken advantage of Fateman's discussion, and was and is a great achievement, (4) that computer algebra is an art that is not self-evident and (5) that science and education are served with an open source computer algebra system of similar quality as *Mathematica*.

Note

This paper is best understood in the context of my book Colignatus (2009a)"Elegance with Substance" on mathematics education - see http://www.dataweb.nl/~cool/Papers/Math/Index.html - and the paper Colignatus (2009b) "Towards Mathix / Math-x, a computer algebra language that can create its own operating system" - see http://www.dataweb.nl/~cool/Papers/Math/2009-08-19-Math-x.pdf (or html).

This document has been created in *Mathematica* 7. This program allows the inclusion of hypertext (clickable but hidden links) and to print / save as PDF but those PDF (either created directly or via printing via PrimoPDF) apparently have the links removed (or they are not shown in Foxit). It is an option to save from *Mathematica* in HTML format, but, curiously, such a file comes as a collection of files and links and not as a single file whence it is not easily transportable. It also defines a fixed style sheet so that it is no longer possible to change the display text size. It is defeat, but it remains the best approach to include all links *verbatim*.

Taking snapshots in Foxit, I took the standard resolution which appears to be too low for this exercise. It looked OK while editting in *Mathematica* but now printing to PDF it appears not OK. I should have tested this with a few snapshots. It will be tedious to take all those snapshots again ... Printing with PrimoPDF caused rather ugly images. Saving to RTF caused *Mathematica* to collapse (likely since there are input cells with (deliberate) syntax errors). But Save As PDF works. If you use the magnifying glass a bit then you can read fairly well. The advantage of a lower resolution of the snapshots is that the file is not as large.

Introduction

A conclusion of the paper on Mathix / Math-x is: "The major consequence is that we must try to settle on a computer algebra language (syntax). We have syntax of *Mathematica*, Maple, Maxima, Python, Axiom and Haskell. The best must be selected, which gives the CAL [Computer Algebra Language] to be made. One relevant idea is that the existing CAs can be ported to CAL." Looking for a proper CAL name I settled on *Bhaskara*, the inventor of the zero and proper positional system for numbers (in the Indian-Arabian-European line).

I have only hands-on experience with *Mathematica* (and Algol, Pascal and Fortran). I am in favour of using "=" for identity / equality, ":=" for direct assignment and "=:" for delayed / lazy assignment, while *Mathematica* uses "==" for identity / equality, "=" for direct assignment and ":=" for delayed / lazy assignment. For me, [*a, b, c*] can be an ordered list and {*a, b, c*} an unordered list, while *Mathematica* uses the last for both though with different Attributes. Thus I am not hung up on the *Mathematica* language (with the distinction between the language and the executable). I am in favour of adequate conversion from the *Mathematica* language to Bhaskara (the CAL to be made).

Given my background and given the goal of arriving at a new computer algebra language, it seems a good idea that I comment on Richard Fateman (1992) "Review of *Mathematica*" - http://www.cs.berkeley.edu/~fateman/papers/mma.review.pdf. Fateman is one of the creators of Macsyma (Maxima's precursor) and thus one of the founding fathers of computer algebra. Macsyma indeed is a source of inspiration of *Mathematica* itself, see e.g. http://en.wikipedia.org/wiki/Symbolic_Manipulation_Program. On his website (page retrieved 2009-8-17) he states (undated, likely after 2002):

"A serious critique of Mathematica appears in *Review of Mathematica* from the J. of Symbolic Computation (13, no. 5, May 1992) in PDF form. Although it is now a decade old (and is a review of a computer program that continues to evolve), I think a careful reading shows that the broad comments are still valid."

Thus, reviewing in 2009 for *Mathematica* 7 what still might be the portent of Fateman's 1992 Review of *Mathematica* 2 will help to clarify some conditions for Bhaskara, the CAL to be made.

PM. Advised reading from Fateman's website are:

● 2001 *criticism of the OpenMath project* http://www.cs.berkeley.edu/~fateman/papers/openmathcrit.pdf. For example "The fact that binding, evaluation, substitution and related concepts have been so mangled by computer algebra systems (for example Maple and Mathematica) should have served as object lessons to the OpenMath designers. Instead of standing on the shoulders of those who have gone before, we are standing on their toes."

● 2000 *Software Fault Prevention by Language Choice: Why C is Not my Favorite Language* http://www.cs.berkeley.edu/~fateman/papers/software.pdf

●1999 *Problem Solving Environments and Symbolic Computation* http://www.cs.berkeley.edu/~fateman/papers/pse.pdf

●1996-99 *Symbolic Mathematics System Evaluators* http://www.cs.berkeley.edu/~fateman/papers/eval.ps. “Providing a context for “all mathematics” without mak-ing that unambiguous underpinning explicit is a recipe tha ultimately leads to dissatisfaction for sophisticated users.” and “Perhaps if there is a lesson to be learned from the activ-ity of the last few decades, it is this: For compute scientists to provide at one fell swoop a natural notation and evalua-tion scheme for al mathematicians and mathematics is both overly ambitious and unnecessary.”

Page 1

We will proceed by photographing the relevant sections in Foxit and discuss these.

A Review of Mathematica

RICHARD J. FATEMAN*
(fateman@cs.Berkeley.EDU)
Computer Science Division, University of California, Berkeley, CA 94720, USA

(Received: 16 November 1990)
(Revised: 16 September 1991)

The Mathematica computer system is reviewed from the perspective of its contributions to symbolic and algebraic computation, as well as its stated goals. Design and implementation issues are discussed.

1 Introduction

The Mathematica[1] computer program is a general system for doing mathematical computation described in Wolfram (1988, 1991). It includes a command language, a programming language, and a calculation environment that is oriented toward symbolic as well as numeric mathematics.

This hasn’t changed fundamentally. Now included are links to the web and other programs, special procedures for fast processing of data and parallel computing, and more sound and interactivity (see the Demonstrations internet page).

The back cover of the manual (Wolfram (1991)) provides excerpts from rave notices like “The importance of [Mathematica] cannot be overlooked ... it so fundamentally alters the mechanics of mathematics.” —The New York Times. *Fortune* says “... it will do, instantaneously, virtually all of applied mathematics ... ” Taubes (1988).

Hype aside, the program is without question interesting to mathematicians, computer scientists, and engineers because of its combination of a number of technologies that have arisen in initially separate contexts—numerical and symbolic mathematics, graphics, and modern user interfaces. The exploitation of the PostScript language for plotting contributed to its natural fit into the package of programs initially released for the NeXT workstation.

It still is interesting, especially when we compare it to what is available elsewhere. Main

commercial competitors are likely Maple and Scientific Workplace.

Not all commentary on Mathematica has been uncritical. For example, reviews in *Science* (Foster & Bau (1989)) and *Notices of the AMS* (Herman (1988), Simon (1990)) compare Mathematica's features, reliability and efficiency to similar programs for algebraic and/or numerical interactive manipulation. Additional commentary on the program includes Hoenig (1990), Vogel (1989). A perceptive book review of the reference manual (first edition) has also appeared (McCurley (1990)). 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`). Such forums provide opportunities for valuable exchanges but, especially as "subscribers" become more numerous, the 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.

*This work has been supported in part by the following: the National Science Foundation under grant numbers CCR-8812843 and CDS-8922788, through the Center for Pure and Applied Mathematics and the Electronics Research Laboratory (ERL) at the 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 ERL; and grants from the IBM Corporation, the State of California MICRO program, and Sun Microsystems.

[1] Mathematica is a trademark of Wolfram Research Inc. (WRI).

Page 2

In order not to keep the reader in suspense, my major conclusion is that Mathematica has many flaws. Some of them are substantial and are unlikely to be repaired because they reflect decisions rather than oversights. The gaps between claims and actuality are substantial. These gaps are not all inherent in the nature of mathematical algorithms or representations since competing commercial programs often provide correct answers where Mathematica fails.

One way of improving the state of the art in automating mathematics is to examine current programs critically; our purpose in this review is, in part, to direct attention to shortcomings and suggest improvements.

2 Preliminaries

This review occasionally assumes the reader has a more-than-casual familiarity with Mathematica, and is certainly no substitute for a primer on the subject. Careful study of the major reference (Wolfram (1988, 1991)) may be an adequate substitute for experience in using the program.

Four themes permeate this review corresponding to areas of technology to which Mathematica potentially could make a contribution:

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

These should be kept in mind as the discussion proceeds, but first a brief historical perspective seems in order.

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 and patron of Charles Babbage, inventor of the "Analytical Engine", suggested such usage in 1844 (see Knuth (1969)). It took over a century for the first actual symbolic computation to be cited in the literature. (see van Hulzen & Calmet (1983) or Barton and Fitch (1972) 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 (The Soft Warehouse (1991)), Maple V (Symbolic Computation Group 1990), and AXIOM, (previously referred to as Scratchpad II (Computer Algebra Group (1988))). These are not necessarily better than the second-generation hold-overs — in particular, Macsyma (Moses (1979), Mathlab Group (1983), Pavelle (1985), Fateman (1989)), provides answers when Mathematica sometimes does not; another second-generation system with a wide following is Reduce 3 (Hearn (1984)). These recent systems were built upon research results plus practical experience in using second-generation systems of the late 1960's, including ALTRAN, CAMAL, PL/I-FORMAC, Mathlab, Scratchpad I, Symbolic Mathematical Laboratory, SAC-1, SIN, and others. An early comparison

Page 3

of some of these systems is still worth reading for background (Barton and Fitch (1972)). Stepping back further in time, the first generation systems of the early-to-mid 1960's included ALPAK, FORMAC, Formula Algol, PM, SAINT, SNOBOL and LISP. There are also numerous other "special purpose" systems such as Schoonschip, Sheep, Trigman and Cayley referenced in Buchberger (1983).

Apparently there was a jungle around 1990 as well, with a survival of the fittest since then. But fitness may not be defined in the most convincing intellectual manner.

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 (Wolfram (1988, 1991)). Indeed, the first edition (p. xvvii) indicates in passing that Mathematica "represents a synthesis of several different kinds of software" including some 16 systems. These are not mentioned in the second edition at all. Algorithm documentation receives similar treatment. If you want to know about the "Risch algorithm" (mentioned on page 528 of the first edition, expurgated from all but the index for the second edition) or how factoring is done, you won't find any information or references.

This is accurate as it is but it likely misjudges the intentions by the author. Wolfram (1991) is already thick as it is and in my opinion it is a splendid book that clearly outlines *what can be done* and *how to use the system.* Apart from the discussion on quality below. There is a distinction between academic publishing and user friendliness. Many users of *Mathematica* may never have a need for a Risch algorithm and thus there is limited need to discuss it. People interested in the history of computer algebra can always ask a CA professor and people interested in the Risch algorithm can always ask WRI support or

some mathematician. Clearly, for the CAL to be designed we want a structure that such documentation is basically open source.

Currently http://mathworld.wolfram.com/RischAlgorithm.html and http://reference.wolfram.com/mathematica/note/SomeNotesOnInternalImplementation.html with the statement “For indefinite integrals, an extended version of the Risch algorithm is used whenever both the integrand and integral can be expressed in terms of elementary functions, exponential integral functions, polylogarithms and other related functions.” Clearly this is not fully explicit.

> Although space limitations prevent us from providing details and full references, it is clear that recent developments in other areas have either inspired or paralleled the facilities in Mathematica. These include user interfaces such as the Macintosh environment or other window systems; alternative mathematics display and manipulation systems such as Mathscribe (Soiffer & Smith (1986)), CaminoReal (Arnon *et al.* (1988)), Theorist (Bonadio (1990)), Milo and FrameMaker (Avitzur (1988)); operating systems and languages with interprocess communication, display technology (PostScript in particular); and programming language ideas (object-oriented programming, pattern matching, functional programming).
>
> An important point here is that Mathematica arrived riding the crest of a wave: a mass market newly-formed from the appearance of vastly improved low-cost computing hardware. That mass market was a large part of what Mathematica's precursors (Macsyma, Reduce, Maple, etc.) lacked. Mathematica became news partly because it was new, and not because it was that much better than its predecessors.
>
> In summary, Mathematica is more evolutionary than revolutionary. Not only does it depend heavily on unacknowleged prior art and technology, it is in many respects not so advanced as older systems.

This piece of history is indeed difficult for me to judge. I started using *Mathematica* in 1993, indeed on a PC, and have had no experience with the other systems mentioned. What I have seen of Macsyma / Maxima is only a short look at the manual to taste the syntax. What is a bit surprising is that the commercial version of Macsyma (still at http://symbolics.com/Macsyma-1.htm) has had the same access to all these resources and did not manage to create the same impact as *Mathematica*. Would it only be marketing ? There seems to be something in this amalgam called *Mathematica* that creates a sense of unity and power that causes its success. A limited success though, since WRI did not proceed to replace Microsoft.

4 Examination of the Objectives

Stephen Wolfram, the principal designer of the system and author of the user documentation (Wolfram (1988, 1991)) for the system, specifies several objectives for Mathematica which are summarized or paraphrased below. This review addresses each of the objectives. Occasionally there will be a 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.

Objectives of Mathematica

- To provide a system for doing interactive symbolic mathematical calculations (interactive Mathematica); (§5)
- To provide a repository for mathematical exposition and education (Notebooks); (§6)

Page 4

- To provide a programming language which unifies ideas from procedural programming, functional programming, rule-based programming, object-oriented programming and constraint-based programming; (§§7, 8)
- To provide facilities for exact symbolic computation and arbitrary-precision numerical computation; (§§9, 10)
- To provide a repository for information on the simplification and manipulation of mathematical functions, polyhedral objects, etc. (Libraries); (§11)
- To provide high-quality plotting from algebraic or discrete computations in a format that can be further manipulated (PostScript); (§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; (§§13, 14)

In the final sections we discuss some general issues about the relationship of Mathematica to mathematics.

This is indeed a sound representation of the stated goals of *Mathematica*, still basically true for 2009, and still basically what we would like to see for Bhaskara (an open source CAL to be developed).

5 The Interactive System

Mathematica is intended to be used primarily as an interactive program supporting the day-to-day computational needs of a mathematician or scientist. For the most part it fits 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 (Avitzur (1988)) or Theorist (Bonadio (1990)), or MathScribe (Soiffer & Smith (1986)) or even the considerably older research system DREAMS (Foster (1984)). Each of these systems allows 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 hand-writing of mathematics for raw input—rather than keying in new text and selecting already-displayed material — has never been exploited successfully in past experimental systems. Experience indicates that a 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 uses of Mathematica, the intuitiveness of the system and the user-visible programming language is very important. The two components are discussed in separate sections on the display (in the next section) and a more extensive subsequent section on the programming language.

This "selection of mathematical expession" now is provided for by "Palettes" with "Placeholders" where variables or other input can be entered. Current *Mathematica* allows a wide range of 2D editting. The criteria of "intuitiveness" and "user-visibility" are still essential. Note that interactivity was a great selling point in 1990, and we still want it, but now compilation returns into attention again.

An advertised novelty in Mathematica is its separation of a front-end "interactive" component from a back-end "computational" component. In practice both parts of the program often run side-by-side in the same computer, but in principle, they could be running on distinct machines. Mathematica was not the first system to try this separation since there were prior experiments with the Maple system, and indeed the current commercial version (Maple V) has such a separation. Even so, the separation in Mathematica has

Page 5

still-unrealized potential. Feedback from mouse-input on plots has been demonstrated on some platforms, but only in recent versions of the program, or on advanced workstations (e.g. Silicon Graphics' systems).

Overall, the notion of (say) a Macintosh front-end attached to a supercomputer back-end sounds better than it really is in practice. Supercomputers generally do not run symbolic mathematics programs particularly well. Running both front- and back-end parts of Mathematica on a fast remote computer works well, assuming there is a "front-front-end" such as an X11 window system to display the graphics. Since such a window system is probably in use anyway, and it can be totally ignorant of and unlicensed for Mathematica, it may be preferable.

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.

In *Mathematica* 7 there is still the distinction between front-end and kernel, though the separate window for the kernel has disappeared. There are commands for parallel computing using the dual cores. I am not familiar with the implications for performance.

6 Notebooks and the Display

Mathematica runs on a variety of machines, and the quality of the user interface varies across a spectrum from the universal but workable ASCII-terminal mode to specially-tuned versions for the Apple Macintosh and NeXT lines of computers.

The most sophisticated interface model provided is called a Notebook. The user types commands as text into an outline processor. That is, there is an option of suppressing details of display of material at lower levels. The computer generates additional text and displays into the outline. The graphics sections can be re-displayed as PostScript and edited. Since the Notebooks can be exchanged between computers, there is a simple technique for reproducing results and building up a library, at least if the Notebooks are properly constructed so as not to conflict in the user's space of objects. Although this

The (interactive) notebook was probably the key attraction of *Mathematica* and the foundation of its success (next to sticking to a good mathematical syntax). The expression "suppressing details of display" is important. The user types a + b and this shows on the input line as a + b (in canonical alphabetical order) but FullForm[a + b] shows that the internal implementation is Plus[a, b]. For output there are three standard formats, OutputForm, StandardForm and TraditionalForm. In fact there are lots of Forms:

?*Form

▼ System`

AccountingForm	FortranForm	PaddedForm	SpaceForm
BaseForm	FullForm	ParentForm	StandardForm
BlankForm	HoldForm	PointForm	StringForm
BoxForm	HorizontalForm	PolynomialForm	StyleForm
CapForm	HornerForm	PrecedenceForm	SyntaxForm
CForm	InputForm	PrintForm	TableForm
ColonForm	JoinForm	PromptForm	TeXForm
ColumnForm	LineForm	RealBlockDiagonalForm	TextForm
DisplayForm	LongForm	RecurringDigitsForm	TraditionalForm
EdgeForm	MathMLForm	RuleForm	TreeForm
EngineeringForm	MatrixForm	ScientificForm	ValueForm
FaceForm	NumberForm	SequenceForm	VerticalForm
FontForm	OutputForm	ShowShortBoxForm	

is undoubtedly a useful approach, in some respects it is not as advanced for modeling of mathematical problem solving as some other programs such as Milo (Avitzur (1988)) or Theorist (Bonadio (1990)). 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. They provide interactive type-setting of mathematical expressions. In addition, Theorist supports animation and rotation of surfaces. Although Mathematica provides these latter facilities on a subset of its platforms, it lacks the pencil-and-paper quality of interaction that these other products offer.

I don't know the other programs referred to but *Mathematica* now allows interactive typesetting with placeholders. Animation and rotation now seem standard, with a Manipulate command that creates Java like applications. The pencil-and-paper interaction may still be an issue. At least *Mathematica* now has a Drawing tool, a bit like Paint. The advantage is that it is available within the *Mathematica* environment, so that you can consider to do more with it than only drawing and rotating.

> Although it is possible to import descriptive material from other programs into a Notebook, including digitized pictures or typeset formulas, the linkage is rather roundabout. Perhaps in the future it will be possible to run a `TeXForm` version of an equation through TEX and put it directly into the notebook. This would certainly create a better environment for Notebooks as an alternative mode of publication of mathematics. In fact, there are nine books cited by Wolfram (Wolfram (1991) page xviii) which describe the use of Mathematica in scientific or educational contexts, and some of them are clearly dependent upon Notebooks as an interface. The design of 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 Notebook front-end to be a hindrance. It may very well be a matter of previous experience and developed preferences.

There is a TeXForm now. *Mathematica* can save in LaTeX and Import[file, "NotebookObject"] generates the notebook again. *Mathematica* now is a powerhouse in translating all kinds of file formats.

Hold[Integrate[Sqrt[x], x]]

$$\text{Hold}\left[\int \sqrt{x}\, dx\right]$$

TeXForm[%]

```
\text{Hold}\left[\int \sqrt{x} \, dx\right]
```

> Incidentally, a Notebook will likely contain only a fragmentary record of computations so it departs somewhat from the tradition of the "laboratory notebook" containing all raw data, experimental results etc. It is more like a showcase.

This is important to always keep in mind. For example, Expand[(a + b)(c + d)] will show the result while the Algebrator (of http://www.softmath.com/Polynomial_long_division.htm) will show all intermediate steps (and explanations if clicked). *Mathematica*'s TracePrint allows to see "intermediate steps" but these are not educational. Eventually, we can imagine that each defined function has different kinds of execution, e.g. fast or with intermediate steps of selected complexity.

Rather than this somewhat uninformative TracePrint some students would want a command Teach:

```
Expand[(a + b) (c + d)] // TracePrint
```

Expand[$(a + b)\,(c + d)$]

Expand

$(a + b)\,(c + d)$

Times

$a + b$

Plus

a

b

$c + d$

Plus

c

d

$a\,c + b\,c + a\,d + b\,d$

$a\,c + a\,d + b\,c + b\,d$

Page 6

7 Programming Language

7.1 Overview

The challenge of making computers truly useful (and perhaps making programmers obsolete) is often couched in terms that make it sound like a programming language issue: All one would need is to create the right syntax and semantics to banish all the problems of applications programming.

That solution is not here yet, but the approaches to the challenge through the years can be characterized as mixtures from three streams:

- The elaborate all-inclusive language (like PL/I, Ada, Common Lisp)
- The extensible language (C, Algol-68, Common Lisp)
- The language oriented to a specific application (PostScript, JCL).

This is a very useful distinction. *Mathematica* would be like Common Lisp and of course provide import & export (and executable links) to such specific applications. *Mathematica* actually is like a next shell upon the operating system. It also depends how

"specific" an application is. Postscript is not only for printers but also for graphics, and soon we are discussing formula editting again.

> Mathematica falls mostly in the first camp in that it has adapted in some way nearly every construction appearing in some general language, and it certainly is elaborate. But it includes some extension techniques, and has certainly packaged together some application-specific subroutines. The evidence to date is that, superficially at least, Mathematica is actually fairly comforting to an experienced programmer—most of the familiar tools, as well as some others, are there. There are generally several ways of writing "equivalent" programs using different programming paradigms. More so than in other languages, different paradigms may differ by orders of magnitude in resource consumption. Since no particular programming style is imposed on the programmer, some programmers will use the numerous syntactic shortcuts to produce "write-only" programs (so called because they become incomprehensible shortly after being written). Indeed, the fact that there are so many variations possible is especially discomforting for a programmer concerned about efficiency. Because there is such limited information available on the internal algorithms and data structures in Mathematica, there is sometimes no alternative to trying various versions of an algorithm and timing them. The fact that details might change is not a good excuse—such information could be part of release notes, for example.

This is expressed sometimes that *Mathematica* is very useful for prototyping, i.e. trying to find the right approach. Once the algorithm has been found then it can be ported to a language that can be compiled (if its fast execution is critical). Current machines however are fast so that compilation might not matter much. Documentation remains a problem. WRI's has a Support Department that does a good job. I haven't had occasion that knowledge about a hidden system routine was performance critical, but I can imagine that WRI provides such information on a need to know and confidential basis (since such use can also give ideas for improvements). However, there is a good case for open source software.

> Mathematica does not have an extensible syntax. It uses all the non-alphabetic symbols of the ASCII character set, and quite a few multi-character symbols. Mathematica's designers did not choose the more conventional wisdom that it may be advisable to leave some characters "for the users" for possible syntax extension. Techniques for such extension are fairly easy to adopt using the parser model used by both Macsyma and Reduce (see Pratt (1973)), and is advocated in Common Lisp, as well. Mathematica builds a complete syntactic box, for good or ill. You can't tinker with it unless you write a new front-end processor.

This is not quite true. This caused me to look in the Wolfram (1991) book again, with nostalgic feelings, and check that $PreRead already existed. Since then, a separate Notation package has been introduced with more flexibility to create your own syntax.

```
?$PreRead
```

$PreRead is a global variable whose value, if set,
is applied to the text or box form of every input
expression before it is fed to *Mathematica*. ≫

> From a mathematician's point of view, J. R. Kudera (Kudera (1988)) 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: when a system deviates substantially from common mathematical notation and semantics as well as from conventional programming, it becomes positively hazardous.

That is true. It is an argument not to have too many programs around that perform the various functions in different ways, but to stimulate the use of a single environment with most flexibility but common methods that will be understood well.

> Nevertheless, programming seems inevitable since the system constructors simply cannot anticipate all needs. In the next several sections we look in more detail at various aspects of Mathematica's approach to the support of programming.

Page 7

> 7.2 Object-Oriented Programming
>
> The programming language specialist may observe that Mathematica's version of this popular supports neither hierarchies nor inheritance. This omission considerably weakens the faithfulness to the notion of object-orientedness (for those who care). Type-based dispatch of operations is nicely integrated linguistically with the pattern matcher, however. What the programmer may think of as a function definition is, in some senses, equivalent to a pattern-match and replacement rule associated with an object, usually the main operator (or `Head`) of the function, but alternatively one of its operands. Thus instead of re-programming some central simplification routine to handle a new user-defined symbol `f[...]`,and its arguments it is possible to associate, in some piecewise fashion, simplification rules with `f` itself. For example `f[x_Integer] := 0 /;x<0` defines simplification of `f` at negative integer values. The approach of using some kind of "local" control based on the operator (e.g. `f`) for simplification is actually fairly common and appears in one of the first algebraic simplification programs, Korsvold (1965).

It is useful to quote Niklaus Wirth (2006) http://www.inf.ethz.ch/personal/wirth/Articles/GoodIdeas_origFig.pdf

> Nevertheless, the careful observer may wonder, where the core of the new paradigm would hide, what was the essential difference to the traditional view of programming. After all, the old cornerstones of procedural programming reappear, albeit embedded in a new terminology: Objects are records, classes are types, methods are procedures, and sending a method is equivalent to calling a procedure. True, records now consist of data fields and, in addition, methods; and true, the feature called *inheritance* allows the construction of heterogeneous data structures, useful also without object-orientation. Was this change of terminology expressing an essential paradigm shift, or was it a vehicle for gaining attention, a "sales trick"?

Anyhow, a 1990 package on OO programming is http://library.wolfram.com/infocenter/MathSource/4455/ and a 1993 package is http://library.wolfram.com/infocenter/Articles/3243/

7.3 Contexts and Information Hiding

Any modern programming language intended for building large systems must provide information-hiding capabilities. Mathematica uses its notion of Contexts for this purpose. The major construction for modular system building appears to resemble packages in Common Lisp, and even 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.

I find *Mathematica*'s construction for Contexts and Packages one of its strong points. For example, a Package creates a Context MyPack` that is exported and a Context MyPack`Private` that hides variables that are not exported. This allows the use of similar names (say, variable x) without fear that the values will get confused. The working environment only gives the relevant routines that are exported and is not immersed in all kinds of intermediates. For routine A[x] the true name is MyPack`A but since it is exported it will show as A. Who investigates the definition of A by using ??A will see the use of long names such as MyPack`Private`x. For that reason, I wrote a small routine ShowPrivate that shows the short names. I put this routine in the package Cool`Context`, and it is an idea to apply it to itself. PM. The routine is quite simple in its output layout and does not reflect on the structure of routines; it might be an idea to access the FrontEnd for that.

ShowPrivate[ShowPrivate]

Cool`Context`Private`

ShowPrivate[x_, item_:"", priv_:"Private`"] for x that can be of Symbol or String (not HoldPattern[expr] or Literal[expr]). Checks whether there exists a private context, enters it, evaluates, and leaves again. The intention is that Private context heads are not shown in output. Alternative item is "??", but it might also be "?" and priv might be "". If item = "" (default) then a list of Strings is put out. Obsolete: For x = HoldPattern[expr] or Literal[expr] output is Information[expr].

```
Attributes[ShowPrivate] = {Protected},
 ShowPrivate[x_, item_:"", priv_:"Private`"] :=
Module[{y, h, allow, res}, allow = {Symbol, String};
 h = Head[x];
 If[ !MemberQ[allow, h], Message[ShowPrivate::notin, allow];
 Return[]];
 y = Context[x];
 If[Head[y] === Context, Message[ShowPrivate::nocon, ""];
 Return[]];
 If[StringPosition[y, "Private`"] =!= {}, Message[ShowPrivate::alrpr];
 If[priv =!= "", Message[ShowPrivate::par3]]];
 y = StringJoin[y, priv];
 If[MemberQ[Contexts[], y], Print[y];
 Begin[y];
 res =
 ShowPrivateServer[y, x, item];
 End[];
 , Message[ShowPrivate::nocon, priv];
 res =
 ShowPrivateServer[h, x, item];
 ];
 TableForm[res]]
```

For example, one might wish to assert the rule 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. In particular, system programs that rely on a certain behavior from Log may be damaged. Therefore 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. We found, for example, such a rule had the effect of breaking the Integrate command.

Yes, you have to know how to use Contexts, and it is important to be aware that changing Protected Symbols will affect their functioning. In this case I wrote a small set of replacement rules that I can apply whenever needed. It may well be that I haven't included various rules that might be included but up to now it has served its purposes.

Log[a b^c]

$\log(a\, b^c)$

% //. LogRule

$\log(a) + c \log(b)$

ShowPrivate["LogRule"]

Cool`Tool`Private`

A list of rules to expand logarithms

LogRule = {Log[(x_)/(y_)] :> Log[x] − Log[y], Log[(x__)*(y_)] :> Log[x] + Log[y], Log[(x_)^(y_

This appears to limit severely the utility of rule-based programming as a technique for adding general information to the system. The programmer has three choices:

- Avoid the use of any of the same function symbols as the system;
- Use patterns only as a front-end to the Mathematica system;

Page 8

- (and/or) Use patterns only as a back-end to the real Mathematica system.

  This front or back-end usage might include converting all `Logs` to `logs` for special simplification, and then converting back to `Logs`.

That is basically true. It is not so that the system can do “everything”. The system does what it does. Tinkering with it creates another system. Thus, in my book “A logic of exceptions” http://www.dataweb.nl/~cool/Papers/ALOE/Index.html I gave a routine to redefine the logical constants Not, And, Or and Implies from a two-valued to a three-valued logic. I have tested the effect only within the context of the book. As soon as is clear what three-valued logic means, within that context, for logic as a subject, the user has two choices: either turning three-valued logic off, and switching back to the system definition, or experiment with how the redefition affects the system. It would be inappropriate to assume that I would have tested the application of three-valued logic for the whole of what *Mathematica* can do. Nowadays we are all familiar with the notion that this is how such programs work, and there may be less a sense of “critique” as in 1990.

Perhaps another example will illustrate the difficulty of this approach: If you teach the system that $x + 1 > x$, then the system does not use this information to compute $\max(x, x+1)$. How can one fix this short of reprogramming the system function `Max` as well as all the proprietary system functions that use some internal version of comparison? By the way, this obvious inequality is not true for all possible x representable in Mathematica; Consider $x = \infty$.

Yes, quite true. The construction of a good System thus is a delicate issue. If there is a change that must be applied system-wide then one would need the source to adapt and recompile.

A subtle point, perhaps not intended to be noticed by the casual fan of Mathematica, is that the rule on the back cover of Wolfram (1988) 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.

This is indeed the combination of FullForm and the extension of rules. Again, replacing Log[a b] with the sum of parts is not necessarily a simplification since one might argue that Log[a b] is simplest. Similarly ! and + have been defined and work within the System. If you want something else to happen then you can define factorial or myFactorial and plus or myPlus. Those own definitions will not affect the System, but if you want that, and there is no adequate $PreRead, then you need the *Mathematica* source code, adapt and recompile.

> It is possible to group rules and apply them together, as illustrated by programs in the on-line library as well as in Maeder (1988). 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.

"Take affect when appropriate" is badly defined. One rule turns Log[a b] into the sum of logs, but another rule does the converse, and if we apply both then nothing changes, or when we apply them repeatedly then we get an infinite loop. The programmer needs to be aware of the final effect and then group his rules such that that result arises. This is a natural consequence of the concept of a replacement rule.

> 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 hidden for proprietary reasons. Even if you are willing to pay the substantial penalty in performance, you cannot tell how much functionality must be recreated.

Quite true. This is what programming is about. The point is rather that *without Mathematica* you actually would have to write your own *Mathematica* anyway, so that the advantage of *Mathematica* is that you can try to profit as much as possible from its environment, indeed by such replacement rules To and From.

> An attempt to make extensive use of Mathematica rules in defining a system using abstract data types is reported by Buchberger (1991) who found it resulted in frustratingly slow computation.
>
> 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 in version 1.2) involves the repeated display of fully-qualified and rather lengthy context names.

Indeed, shadowing of names is to be avoided. It is better to use the CleanSlate` package for garbage collection before loading the packages that you need. For this reason I wrote my own ResetAll routine that either applies CleanSlate` or loads it. Debugging is always

an issue. Since *Mathematica* is so well structured (and I program rather structured) debugging is less of a problem. One generally assumes that packages written by others have been tested and are less relevant for debugging. Thus debugging concentrates on the programs that you write yourself. But then you have a great flexibility of testing parts in the Global` context. Indeed, the lengthy context names: see ShowPrivate above.

7.4 Spaces mean Multiplication

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 `2 x` 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 (Computer Algebra Group (1988)), following

Scratchpad now is open source axiom, at http://www.axiom-developer.org.

Page 9

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: `Sin[x]`. This departure from conventional notation may be of special concern to a teacher whose students may be struggling with notation in the first place. There is a weak argument that most potential users have not had experience "typing" conventional mathematics in *any* notation, and therefore requiring square-brackets may not be objectionable.

I discuss the use of brackets in my book "Elegance with Substance" http://www.dataweb.nl/~cool/Papers/Math/Index.html. Students can have problems with notation also with sin(*x*). As some textbooks give $a(b + c) = ab + ac$ then sin(*x*) might turn into sin *x.* Possibly the *italics* cause warning flags. In general I am greatly in favour of the capitals for the system functions. Sin and Cos are names like John and Mary. CompoundWords read better when the first letter is upper-case as well. Textbooks don't do so but they ought to adopt the practice. Not for the reason that you can define your own sin and cos but for these other reasons. In my own programs my own function definitions are also with CompoundWords. There is no need for distinction from System functions while there is all the reason to have uniformity in reading syntax.

Also, we can imagine that the distinction between (and [was best for programming parsers in 1990, or earlier indeed, but nowadays we can be a bit more ambitious. For reading it is pleasant to have different brackets merely for bracketting. With y * [x - (a + b)(a + c)] you can have an easier discussion with your students whether you start with the round or the square brackets. Since the comma is a special case, it can be recognized as

an exception and we still can allow freedom of use, depending upon preferences or readability, for (a, ..., b) = [a, ..., b] to represent an ordered list, and *f*(a, ..., b) = *f*[a, ..., b] to be a “tagged ordered list” or function call. To prevent confusion about the single valued function call *f*(a) = *f*[a] we only need to require that textbooks stop writing *a*(*b* + *c*) = *ab* + *ac* and instead properly write *a* (*b* + *c*) = *a b* + *a c* with sufficient blanks inserted.

Many programming languages have the convention to write *function parm1 parm2* with blanks instead of brackets *function*(*parm1, parm2*). This can better be discouraged. Those languages can flourish in niches of specialised applications. It may allow for simple parsing, without testing for brackets and commas. However, it is better to have a common syntax, for readability, ease of learning, porting, integration of possible applications.

Thus it indeed would seem to be possible to write multiplication with blanks - though retain *x* * *y* = *x y* since sometimes we need this expressiveness and flexibility.

PM. From Fateman (1990) I take the example of a! !b that should be multiplication a! * !b but that still is parsed wrongly as (a!)! * b, given the “problem” that a!! is not repeated factorial but a single Factorial2. One supposes that once a parsing error is made then this becomes part of official syntax ...

a! // FullForm

Factorial[*a*]

a!! // FullForm

Factorial2[*a*]

! b

¬ *b*

a! ! b

b (*a* !)!

% // FullForm

Times[*b*, Factorial[Factorial[*a*]]]

(a!) (! b)

a ! ¬ *b*

The use of // such that *x* // Sin = Sin[*x*] is inadvisable. See “Elegance with Substance” for dynamic division such that *x* // *x* = 1 after simplification with the defined agreement that

this also holds for $x = 0$ (which is not the case for x / x).

> 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 parses this expression as the necessarily meaningless `(a++)++ * b`. Even the author of the output-printing program was confused on this, since echoing back the first expression (by typing `Hold[a++ * ++b]`) displays the non-equivalent `Hold[a++ ++b]`. This particular inconsistency of the display with the internal form was reported as a bug some time ago, but its existence suggests a design error in that the parser and display should not have access to inconsistent precedences. In a well-designed system the two closely related subsystems would use the same precedence data, stored in one place.

I avoid this kind of syntax and only report that the bug still exists or is no bug.

Hold[a++ * ++b]

Hold[*a*++ ++*b*]

ReleaseHold[%]

Increment::rvalue :

a is not a variable with a value, so its value cannot be changed. ≫

PreIncrement::rvalue :

b is not a variable with a value, so its value cannot be changed. ≫

a++ ++*b*

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

3! ++ a

Set::write : Tag Factorial in 3! is Protected. ≫

6 *a*

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

Physicists should not write 1 / 2π when they mean 1 / (2 π).

> Finally, the accidental omission of a semicolon or comma will not be caught by a syntax check. The forms `f[a b]`, `f[a,b]` and even `f[a;b]` are all unexceptional syntactically.

Well, these are different expressions. So type well. An accidental omission or inclusion of a 0 in 1000 is neither easily caught. A check whether you have a driver's license is not the same as testing whether you will use it wisely today. Here, *f*[*a; b*] means that first *a* is evaluated, then *b* and then *f*[*b*]. If *b* depends upon *a* then we get *f*[*b*[*a*]]. All this is syntax and not an error of the syntax. If you are likely to make those errors or when it is critical

that you don't make these errors, write a routine to catch you doing so. PM. When writing a program, then sometimes it can be handy to use {a, ..., b} with commas as delimiters instead of semicolons, since this allows you to see the intermediate evaluations. For example:

```
{{aa = 3 / 4^2; bb = Sqrt[aa]}, versus, {aa = 3 / 4^2,  bb = Sqrt[aa]}}
```

$$\left\{\left\{\frac{\sqrt{3}}{4}\right\}, \text{versus}, \left\{\frac{3}{16}, \frac{\sqrt{3}}{4}\right\}\right\}$$

My conclusion is that insisting on the use of an asterisk is preferable, because leaving it out brings up too many problems. One reader suggests as an alternative inserting parentheses when there is any doubt. This advice is hardly likely to be followed by those who need it.

The argument is not that strong. My impression and experience is that it is pleasant to have both blank or asterisk available to express multiplication, so that you are free to choose depending upon the situation. For example y * [x - (a + b)(a + c)] as used above is slightly smarter to use than y [x - (a + b)(a + c)] since the latter might almost be a function call. But y * [x - (a + b) * (a + c)] is needlessly pedantic.

7.5 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 a collection of arguments. (If `f[x__]` is matched to `f[a,b,c]` then `x` is `Sequence[a,b,c]`) 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 mistyped `f(a,b)` is somewhat unexpectedly mapped into `a*b*f`. (Explanation: `f(a,b) = f*(a,b) = Times[f, Sequence [a,b]] = Times[f,a,b]`). If you type `f(0,x)` you get 0. A simple fix for this problem is to be less clever and forbid the alternative input-syntax of `(a,b)` for `Sequence[a,b]`. Version 2.0 has adopted this fix. You still should beware that `f(0)` is 0, regardless of the definition of `f`. Note that the `Head` function does not work on a `Sequence` since `Head[Sequence[a,b]]` is equivalent to `Head[a,b]`, which is meaningless.

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. Here you need the space:

Sequence[a, ..., b] is an extremely useful object. As said, my suggestion is that (a, ..., b) = [a, ..., b] is an ordered list, as distinct from unordered set {a, ..., b} and Sequence[a, ..., b]. *Mathematica* currently however has (a, b) as *undefined* and requires that we change the input.

```
(a, b)
```

```
f@(a, b)
```

```
f(a, b)
```

Sequence[a, b]

Sequence[*a*, *b*]

(Sequence[a, b])

Sequence[*a*, *b*]

[a, b]

[Sequence[a, b]]

{Sequence[a, b]}

{*a*, *b*}

Note that Sequence[a, ..., b] differs from [a, ..., b]. A sequence disappears into another sequence but an ordered list remains an object.

{f[a, b, Sequence[c, d]], versus, f[a, b, [c, d]]}

{*f*(*a*, *b*, *c*, *d*), versus, *f*(*a*, *b*, [*c*,] *d*)}

Indeed f(0) is multiplication but we advised that this is avoided.

f (0)

0

While we do not Unprotect Factorial, *Mathematica* now recognizes Unequal and automatically replaces it.

big ≠ bigfact

big ≠ bigfact

Page 10

```
big! =bigfact
```

because the language includes a not-equals operator (`!=`). Finally, the expression

```
4/. 4->5
```

which means "substitute 5 for 4 in the expression 4," returns `5`. It has a rather different meaning from the expression without the space in it;

```
4/.4->5
```

which returns the rule `10.->4` because `4/.4` is `10.` What's the point in all this? Simply that it is potentially quite confusing to see a modern programming language design in which the meaning of "white space" is not only significant but has puzzlingly different meanings depending on context.

(Typing error: 10. → 4 must have 5.) I wonder whether it really is an issue of white space and not an issue of the decimal dot ... For example: Typing "1.2.3" used to be ambiguous but now the input interface inserts a white space to show how it will be sparsed.

1.2 .3

0.36

While this of course gives the Dot operation:

a.b.c

a.b.c

% /. Thread[{a, b, c} → {1, 2, 3}]

1.2.3

But given the priority of symbols, the original interpretation still stands.

4/.4 → 5

10. → 5

It doesn't seem much of a problem. We might use /* instead of /. but then a critique might become that 4/*4–5 is sensitive to typing errors, as for example you might have intended to type 4/2*4–5 and accidently omitted the 2.

7.6 Procedures, Functions, Patterns, Efficiency

There are attractive aspects to the method used to define procedures. Were it not for the difficulties caused by errors in semantics, and the almost inevitable inefficiency which results from the reliance on matching, it would be even more attractive. In a nutshell, the approach is to

- Incorporate control structures from all of C, Lisp, APL and functional programming.
- Unify the three notions: a mathematical function $f(x)$; a pattern `f[x_]`; and a procedure invocation. Thus $f(3)$ is "implemented" by a pattern match of `f[3]` which results in a temporary binding of x to 3. The evaluation of the right-side of a rule provides the semantics for the function. (Mathematica has other ways of viewing function objects meant as programs, based on the lambda calculus and Lisp. You can express $\lambda(x, y).(x + y)$ as `Function[Plus[Slot[1], Slot[2]]]` or, in the runic syntax `#1+#2&`.)

Unfortunately many difficulties with details crop up.

7.6.1 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 `x_ - x_ := 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 would be an error to simplify `Infinity - Infinity` to 0. Indeed, Mathematica returns `Indeterminate`. And yet to Mathematica, the expressions `f[Infinity]-f[Infinity]`, `f[RealInterval[{0,1}]]-f[RealInterval[{0,1}]]` and `Random[Integer,n]-Random[Integer,n]` are each 0 (although the latter provokes an error message).

PM. This proposed rule cannot be accepted and we would have to use :> (delayed rule):

```
x_ – x_ := 0
```

SetDelayed :: write : Tag Plus in −x_ + x_ is Protected . ≫

$Failed

The other points still apply. The reason is that the expressions are taken to be formal objects. As long as f and n are undefined then the evaluator might either return the same expression (as unevaluated) or it can take the formal objects and conclude that Object - Object gives 0. It is a choice. We can define Plus[Object, -Object] to generate 0 and PlusAlt[Object, -Object] to remain as it is. A key point is to be aware of it. Actually, for most applications the 0 answer will be most useful. But of course, these more exceptional cases might cause a surprise if you have not been aware of them.

```
f[Infinity] – f[Infinity]
```

0

```
Random[Integer, n] – Random[Integer, n]
```

Random::randn :

Range specification n in Random[Integer, n] is not a valid number or pair of numbers. ≫

Random::randn :

Range specification n in Random[Integer, n] is not a valid number or pair of numbers. ≫

0

> Similarly, two occurrences of $O(x^2)$ are not semantically identical because each may imply a different asymptotic constant. In particular, $O(x^2) - O(x^2) = O(x^2)$ would seem most plausible. 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 (Graham *et al.* (1989)).
>
> The problem here is fairly deep, and quite important. How could one modify the rule that `x_ - x_ :=0` to make it correct? Perhaps by saying that `x` must satisfy some

Note the somewhat peculiar notation switch between input and output:

```
O[x]^2
```

$O(x^2)$

Apparently this works OK:

```
O[x]^2 – O[x]^2
```

$O(x^2)$

Page 11

> 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, one 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 among the "dangerous" kind indicated above. 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 (see Computer Algebra Group (1988)) or Newspeak (Foderaro (1983)).

I wonder whether Scratchpad and Newspeak really solved the "problem" since it indeed is a "deep" problem in the sense that there is the fundamental choice what to do with undefined symbols.

In principle, all System symbols are defined so that one might formulate conditions on x_ - x_ :> 0 using what follows from the definition. The problem are the undefined symbols that the user generates. Since all System and known package symbols generally are Protected, I wrote the NonAttributedSymbols routine to find the user symbols - and using this knowledge the user might introduce a $PreRead to warn about occurrences of x - x that are not properly defined.

```
?NonAttributedSymbols
```

```
NonAttributedSymbols[expr] gives the
    symbols x in expr for which Attributes[x] === {}
```

```
NonAttributedSymbols[Hold[f[Infinity] – f[Infinity]]]
```

```
Attributes::ssle :
  Symbol, string, or HoldPattern[symbol] expected at position
      1 in Attributes[∞]. ≫
```

```
{f}
```

```
NonAttributedSymbols[Hold[Random[Integer, n] – Random[Integer, n]]]
```

```
{n}
```

But is it really a fundamental problem by itself ? What about this:

```
f[Infinity] / f[Infinity]
```

```
1
```

There probably is a multitude of possible formal expressions that might require such checking and warning, at least, if you follow that route. It is analytically simpler to know that Objects will be taken as identical as long as they are undefined. If potential semantics require a different approach then it is up to the user who entertains that semantics.

> Mathematica encourages the user to define functions by writing collections of transformation rules, each of which has a pattern, a replacement, and optionally, some conditions. This mechanism has a long history as a model of computation, and each formalism in the past has had to define the order in which rules are applied. This order has a strong bearing on the real semantics of a rule set, since different orderings can change the answers or even cause an infinite loop.
>
> How are Mathematica's rules ordered?
>
> The manual explains that the most explicit rules are used in preference to the most general. Yet it is clear that except for very simple cases, Mathematica has not got the slightest clue as to which rules are more specific. Maeder (1988) p. 59 points out "...Mathematica cannot always find out which of the rules is more special than the other and it might fail to reorder them accordingly." If the rules are not ordered correctly, it may be fairly painful to write patterns to make sure that the pre-conditions for matching do not overlap. Furthermore, if you "cover up" a built-in operator with rules, and then you wish to refer to the built-in routine, you have no direct access to the system's prior definition. In version 2.0, WRI apparently gave up on getting the ordering right and the user is given a chance to rearrange the rules by setting (for example) `DownValues` explicitly (see p. 266 of the 2nd edition of the manual). It is implausible that users would find this a convenient way of "correcting" a rule set.

Yes, quite true. In practice, when there are few rules you know what to expect but when there are more rules then the order is relevant, and as a user you don't know either the rules nor their order. So you build up an experience as to what to expect. It will be useful when the rule-book is made open source and that users can adapt and recompile.

In many cases it is easier to build up that experience and act from there then studying and adapting such a rule-book, since the latter is likely to be complicated. My book on logic, ALOE, Colignatus (2007), uses meta-formatted axioms for first order logic and a generator to create all the combinations for actual replacement. Then the user must check this process or the long list of created rules. It is optional to change the order, but only to a limited extent (kinds of rules are created together and thus bunch up). The main point in the discussion in the book is that transforming logical expressions by means of replacement rules is tricky. I can usefully quote:

Quote

It turns out that Infer is not without a paradox. We can trace this paradox to the Ex Falso Sequitur Quodlibet situation, or, that from a falsehood anything can be derived. It is useful to be aware of this, for otherwise we might conclude that there is an error in our InferenceMachine. The following is a crucial example.

- The following applies the transitivity of If:

  (x ⇒ !x) ∧ (!x ⇒ x) /. Infer

  $(x \Rightarrow x)$

% /. Infer

True

- While SelfImplication gives (now using //. rather than /.):

(x ⇒ !x) ∧ (!x ⇒ x) //. SelfImplication

(¬ *x* ∧ *x*)

% //. InferAndOr

False

Unquote

Page 12 - 32

This discussion becomes too long. Fateman (1992) already is a sizeable discussion and comments on it doubles the size of this paper. It suffices to stop here for the kind of conclusions that I want to draw. However, there is this small exception:

> 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. The competitors to Mathematica are not without fault; they tend, however, to be more cautious. For example, Macsyma checks to see if $n = -1$ before returning the value $x^{n+1}/(n+1)$ for $\int x^n dx$. Mathematica does not. DERIVE finesses this problem by returning $(x^{n+1} - 1)/(n+1)$ which yields the correct limit as $n \to -1$.

This indeed gives a problem:

Integrate[x^n, x]

$$\frac{x^{n+1}}{n+1}$$

% /. n → –1

Power::infy : Infinite expression $\frac{1}{0}$ *encountered.* ≫

ComplexInfinity

For pure numbers there is no problem:

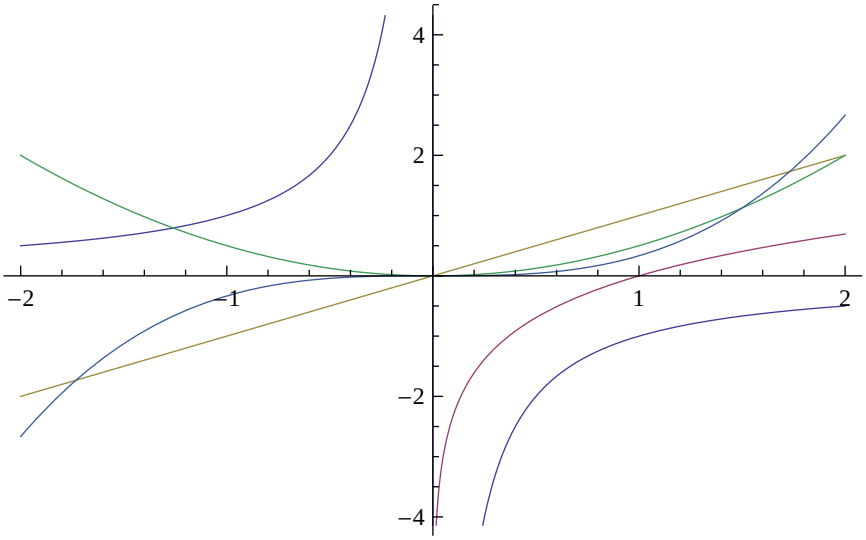
Table[x^n, {n, –2, 2}]

$$\left\{\frac{1}{x^2}, \frac{1}{x}, 1, x, x^2\right\}$$

Integrate[#, x] & /@ %

$$\left\{-\frac{1}{x}, \log(x), x, \frac{x^2}{2}, \frac{x^3}{3}\right\}$$

Plot[%, {x, –2, 2}]



This would be like derive:

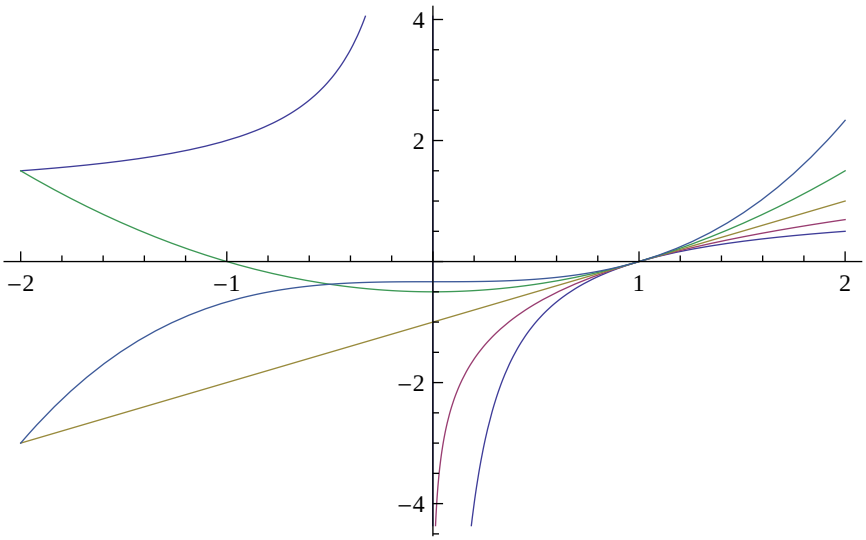
Integrate[x^n, x] – 1 / (n + 1)

$$\frac{x^{n+1}}{n+1} - \frac{1}{n+1}$$

Table[Limit[%, n → m], {m, –2, 2}]

$$\left\{\frac{x-1}{x}, \log(x), x-1, \frac{1}{2}\left(x^2-1\right), \frac{1}{3}\left(x^3-1\right)\right\}$$

Plot[%, {x, –2, 2}]



PM. See however Colignatus (2009:91) for the redefinition of calculus by using dynamic division // instead of limits. Log[*x*] is just shorthand for writing {(*x*^*h* - 1) // *h*, set *h* = 0}, and in the above *h* = *n*+1. The result in Derive is proper, even though there is an argument that 1 / (*n*+1) is only an integration constant. It is not quite clear to me why the choice of standard constant is determined by requiring the same outcome for *x* = 1 except that it looks prettier and allows consistency with the Log, but there may be a deeper reason.

Page 33, Fateman's Conclusions

17 Conclusions

This review is hardly the final word on Mathematica, a program we expect to continue to change even as it has changed substantially during the writing of this commentary. In fact, we hope that some improvements in the program were prompted by earlier drafts of this paper. In the interests of brevity, almost all mentions of earlier bugs now fixed have been dropped. In a few places some discussion of features of "obsolete" versions of the system remain for the following reasons:

- In some sense, every user has an "old" version. The as-yet-unreleased version (as we write, 2.0 is not yet generally available) cannot be reviewed satisfactorily.
- 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 indicated here, as well as other criticisms. Alternatively, or in addition, commercial or academic "rivals" may provide new or better solutions to these problems.

For those persons waiting eagerly for mathematicians to be replaced by a universal computer program that "does mathematics", it is our opinion that this will require the development of technology that does not yet exist. Continuing research should learn from the Mathematica experience in combining symbolic mathematics in a general scientific information and programming environment with applications for research and development, teaching, and even entertainment.

My conclusions

We have skipped page 12-32. Personally I would like to look deeper into them before commenting. For what we have seen the discussion already was enlightening.

My hope is that the reader comes away with the notions: (1) that Fateman provides a highly valuable review of *Mathematica* 2 and general points in computer algebra, still relevant for *Mathematica* 7, (2) that some points may be a bit overdone but most points are on target, (3) that *Mathematica* 7 has taken advantage of Fateman's discussion, and was and is a great achievement, (4) that computer algebra is an art that is not self-evident and (5) that science and education are served with an open source computer algebra system of similar quality as *Mathematica*.

References

Colignatus, Th. (2007), “A Logic of Exceptions”, Thomas Cool Consultancy & Econometrics, http://www.dataweb.nl/~cool/Papers/ALOE/Index.html

Colignatus, Th. (2009a), “Elegance with Substance. Mathematics and its education designed for Ladies and Gentlemen”, Dutch University Press, http://www.dataweb.nl/~cool/Papers/Math/Index.html

Colignatus, Th. (2009b), “Towards Mathix / Math-x, a computer algebra language that can create its own operating system” - see http://www.dataweb.nl/~cool/Papers/Math/2009-08-19-Math-x.pdf (or html)

Fateman, R. (1992), “Review of Mathematica”, revision 1991, from the J. of Symbolic Computation (13, no. 5, May 1992), http://www.cs.berkeley.edu/~fateman/papers/mma.review.pdf. PM. There is also a 1993 review, see http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.9348

Fateman, R. (1990), “A Lisp-language Mathematica-to-Lisp Translator”, SIGSAM Bulletin 24, no. 2 p 19-21 (April, 1990) and Computer Algebra Nederland Nieuwsbrief 6, October, 1990, http://www.cs.berkeley.edu/~fateman/papers/lmath.ps

Wirth, N. (2006), “Good Ideas – Through the Looking Glass”, IEEE Computer, Jan. 2006, pp. 56 – 68, http://www.inf.ethz.ch/personal/wirth/Articles/GoodIdeas_origFig.pdf

Wolfram, S. (1991), “*Mathematica*: A System for Doing Mathematics by Computer”, Second Edition, Addison-Wesley

Wolfram Research Inc. (2008), *Mathematica* 7.0.0, see http://www.wolfram.com