NOTICES OF THE AMERICAN MATHEMATICAL SOCIETY

ARTICLES

FEATURE COLUMNS

DEPARTMENTS

Computers and Mathematics

Edited by Keith Devlin

This month's column

Three software reviews make up the first column of 1994. First, Larry Lambe looks at *AXIOM*. Then, Suzanne Molnar reports her experiences with the Student Edition of *Object Logo*. Finally, Jim Northrup reviews *Fields&Operators*.

All three reviewers have contributed to the column in the past, and it is good to see them back. But I am always on the lookout for new reviewers. In particular, my list of volunteers willing to review Macintosh software is starting to run down. If you use a Macintosh and would like to make your own contribution to the column, please send me a message at the address below (e-mail or snail-mail), mentioning any particular preferences as to the kind of software you would like to review.

Editor's address:

Professor Keith Devlin
School of Science
Saint Mary's College of California
P.O. Box 3517
Moraga, California 94575

Correspondence by electronic mail is preferred, to:

`devlin@msri.org`.

Reviews of Mathematical Software

AXIOM System

Reviewed by Larry Lambe*

A little more than four years ago, I wrote about "*Scratchpad* II as a tool for mathematical research" in this column **[L1]**. *Scratchpad* has grown into what is now called the *AXIOM* system, and there is a lot to say about this evolution. I think of it more as a maturation, although neither term is quite right for what has happened. In fact, as you will see, my original remarks about the mathematical nature of *Scratchpad* can be taken verbatim in connection with *AXIOM*. Some of the new things are a user interface that rivals anything that can be found in the market these days, a flexible graphics interface that can provide both insight and enjoyment, and a new book **[JS]** that covers the system quite nicely.

Specifically, *AXIOM* is a "mathematically object-oriented" environment consisting of five major components and a sixth that is under development and soon to be released. They are

1. an interactive computational environment,
2. a "hypertext" interactive documentation system that is user programmable,
3. a graphics package that manipulates and displays objects in two and three dimensions,
4. an object-oriented language,
5. an extensive mathematical library compiled into machine code for efficiency with complete access to the source code for all users, and
6. a link to external libraries written in other languages.

The thrust of the 1989 article was the object-oriented nature of the system and, in particular, its inclination towards mathematics. This is an important and distinguishing feature of *AXIOM*. Issues such as "code reusability" have been around in computer science for some thirty years. The notion of parameterized types in the formal theory of computer languages goes back quite a way as well. These days more and more of such concepts are finding their way into other areas of science that use computer aids.

We are still in a time when there are different terminologies in use for exactly the same concepts in different object-oriented languages. Because of this, it will be useful to set up a dictionary, through the use of analogy, to define some terms.

I'm pretty sure that you've all heard phrases like "object-oriented thinking" from other sources. I will not attempt a definition here, but since I am addressing mathematicians, I can safely say that you should be familiar with it, since most of you do it. In object-oriented programming, however, there are also some important ideas needed that fall outside of traditional mathematical experience. The best way to proceed is to think about the foundations for some of the usual structures we encounter in mathematics, for example, polynomials.

Let's write the free module on a set `X` over a ring `R` as `FreeModule(R,X)`. Of course, in mathematics, there is no

*Larry Lambe is at Rutgers University in New Jersey. He can be reached by e-mail at: `llambe@cesl.rutgers.edu`.

The author is grateful to members of the Computer Algebra Group, IBM Research, Charlie Fletcher, Philip Santas, and Nicolas Robidoux for useful conversations concerning this note.

trouble in realizing a functor such as "FreeModule", whose parameters (arguments) are a ring R and a set X and whose value is a module over R, as a concrete object.

Given the functor above, we can easily define all sorts of mathematical structures. For example, if we are given a monoid M, i.e., a set M with a binary operation $*:M\times M\to M$ which is associative and has an identity element, we can form the monoid ring of M over a ring R by defining an operation on FreeModule(R,M) as follows. First define a function

$$\mathtt{M}\times\mathtt{M}\to\mathtt{FreeModule(R,M)}$$

by simply "coextending" the given operation on M. Now extend this function bilinearly to

$$\mathtt{FreeModule(R,M)}\times\mathtt{FreeModule(R,M)}\to\mathtt{FreeModule(R,M)}.$$

This gives a mathematical structure which we will denote by MonoidRing(R,M). Let's agree to call these functors "constructors" to emphasize the point that they produce new mathematical objects out of collections of others. It is now easy to see that we can get an object isomorphic to the usual polynomial ring in one indeterminant over a ring R by simply forming MonoidRing(R,$\mathbb{N}$), where $\mathbb{N}$ is the monoid of natural numbers with addition. If we agree to write a linear combination $r_1n_1 + \cdots + r_kn_k$ where $r_i \in R$ and $n_i \in \mathbb{N}$ as $r_1t^{n_1} + \cdots + r_kt^{n_k}$, we obtain the usual representation of polynomials as well.

It might surprise you to find out that the polynomial ring in one indeterminant over an arbitrary ring R may be defined in *AXIOM* in *exactly* the above way. Furthermore, there are facilities for providing a wide range of display forms automatically (so elements of MonoidRing(R,$\mathbb{N}$) can indeed be made to display as polynomials in "t").

Two important components of object-oriented paradigms are *encapsulation* and *inheritance*. In *AXIOM*, an abstract datatype has the properties of encapsulation (private and public parts, etc.). Datatypes in *AXIOM* are typically parameterized and represent mathematical structures. An important consequence of the object-oriented paradigm (in the above sense) is *polymorphism*, i.e., objects (programs and mathematical structures) can be reused in a variety of contexts. The abstract type FreeModule(R,X) is parameterized by the abstract types Ring (the R parameter) and Set (the X parameter). Furthermore, note that the addition in MonoidRing(R,M) comes from its "parent" FreeModule(R,M) upon which it is built. This is an example of inheritance. In this light it is clear that these aspects of object orientation have been present in mathematics for quite some time.

Concepts falling outside of the traditional mathematical experience, but relevant in a discussion of object-oriented methods, are the notions of dynamic binding and dynamic dispatch as well as dynamic memory allocation and automatic garbage collection. We will not go into detail concerning these concepts here, but the interested reader will find more information in the references [C], [MW]. Object-oriented languages do not have to have built-in memory management. C++ is an example.

None of the major computer algebra systems today have parameterized types built into the language except *AXIOM*. On the other hand, all of the major computer algebra systems have some form of dynamic allocation and automatic garbage collection built in. It is fair to say that these latter concepts are what make symbolic computation systems so attractive to most researchers. Without them a user is not free to spend all of his time concentrating on mathematical concepts. Instead, he or she must, for example, constantly make sure that there is enough memory available for a process which may be growing in a way that is not measurable before execution and also come up with some scheme for reclaiming memory that has been used, but which will not be used again unless steps are taken to make it so. Let me now go on to say some specific things about the six components of the *AXIOM* system given above.

First, there is the interactive environment. Among computer algebra systems, *AXIOM* is unique in the way that it dynamically builds datatypes based on user input. If you enter $x**2 + 1/3$, it will build polynomials with rational coefficients. If you enter $x**2 + 0.333*\%i$, it will create polynomials with complex coefficients. Type inferencing also applies to function definitions. You can define a function f by $f(x) == x**2$. If f is applied to an integer, the type of f is chosen to be Integer → integer. If f is applied to a rational function such as $1/(x+1)$, the type of f is chosen to be

Fraction Polynomial Integer → Fraction Polynomial Integer,

etc.

Occasionally, type declarations are necessary. *AXIOM* provides for that. For example, to declare x to be a polynomial with integer coefficients you may use the syntax x:POLY INT. In fact, all of the choices *AXIOM* makes can be made instead by the user, if desired.

The hypertext facility is called "HyperDoc" in *AXIOM*. A sequence of windows is displayed on the next page. The windows should be read from left to right and top to bottom. Beginning with the "HyperDoc" window, the next window was obtained by clicking on "Basic Commands". The "Series" field was clicked on to give the third window, "Series Basic Command", and in that window the choice for "Formula" was chosen. This produced the fourth window, "Power Series Basic Command", in which "Puiseux Series" was chosen. That produced the fifth window. At this point some other choices can be made. It is possible to overwrite the data which automatically come up in the "Puiseux Series Basic Command" window and enter other data. This makes it convenient to experiment with *AXIOM*. The given data were chosen. By clicking on the "Continue" button, those data were used to create a valid *AXIOM* statement displayed in a new window labelled "Basic Command". Some other HyperDoc pages cause collections of statements to be generated. If the "Do It" button is clicked, the statement is executed in the

EXIT HELP — HyperDoc

This is the top level of HyperDoc. To select an item, move the cursor with the mouse to a word in **this font** then click a mouse button. For an introduction to HyperDoc, click on **HELP**.

What would you like to do?

Basic Commands	Solve problems by filling in templates.
Topics	Learn how to use Axiom, by topic.
Browse	Browse through the Axiom library.
Examples	See examples of use of the library.
Reference	Scan on-line documentation on Axiom.
Settings	Axiom system commands and variables.
HyperDoc	Write your own HyperDoc.

EXIT — Basic Commands

Calculus	Compute integrals, derivatives, or limits
Matrix	Create a matrix
Draw	Create 2D or 3D plots.
Series	Create a power series
Solve	Solve an equation or system of equations.

EXIT — Series Basic Command — **HOME**

Create a series by:

Expansion	Expand a function in a series around a point
Formula	Give a formula for the *i*'th coefficient

EXIT — Power Series Basic Command — **HOME**

Select the kind of power series you want to create:

Taylor Series
Series where the exponent ranges over the integers from a *non-negative integer* value to plus infinity by an arbitrary *positive integer* step size

Laurent Series
Series where the exponent ranges from an arbitrary *integer* value to plus infinity by an arbitrary *positive integer* step size

Puiseux Series
Series where the exponent ranges from an arbitrary *rational value* to plus infinity by an arbitrary *positive rational number* step size

EXIT — Puiseux Series Basic Command — **HOME**

- Enter the *formula* for the general coefficient of the series
  `(-1)**((3*n - 4)/6)/factorial(n - 1/3)_`
- Enter the *index variable* for your formula: n
- Enter the *power series variable*: x
- Enter the *point* about which you want to expand: 0

For Puiseux Series, the exponent of the power series variable ranges from an *initial value*, an arbitary rational number, to plus infinity; the *step size* is an any positive rational number.

- Enter the *initial value* of index (a rational number): 4/3
- Enter the *step size* (a positive rational number): 2

Continue

EXIT — Basic Command — **HOME**

Here is the Axiom command you could have issued to compute this result:

```
series(n +-> (-1)**((3*n - 4)/6)/factorial(n - 1/3),x =
0,4/3..,2)
```

Do It

Select Exit *to make this window go away.*

original *AXIOM* interpreter automatically. Here is the result:

```
(1) ->series(n +->(-1)**((3*n - 4)/6)/
            factorial(n - 1/3),x = 0,4/3..,2)

        4      10
        -      --
        3   1  3       5
   (1)  x  - - x   + O(x )
            6
Type:
UnivariatePuiseuxSeries(Expression Integer,x,0).
```

There are also facilities for causing new interpreter windows to pop up and execute *AXIOM* commands automatically (e.g., using the "Examples" field of the HyperDoc window). Also, by clicking on the "HyperDoc" field of the HyperDoc window, you can learn how to write your own HyperDoc lessons on any subject you like along the lines of what has been explained (and more).

The system has a convenient browser that lets you find out about a domain's operations, attributes, ancestors in the hierarchy, and cross references. All of this is HyperDoc oriented.

There are tutorials in HyperDoc that cover the basic graphics. The first procedure that I will describe is the "draw" function. This function can be used quite naturally and simply, as in the command

```
draw(x**2,x=-1..1)
```

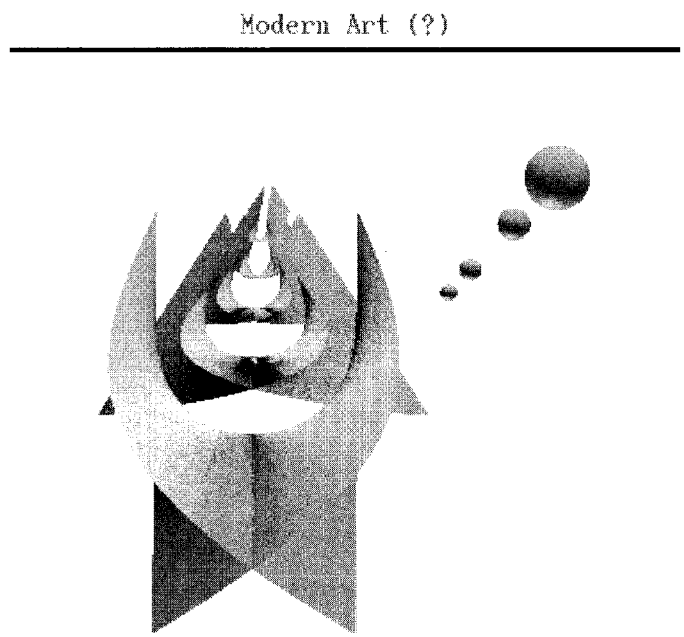
which causes a window to pop up with the graph of the given parabola in the given range. It can, however, also be embellished somewhat, as in the command

```
draw(tan x,x=-2*%pi..2*%pi,
    clip==true,curveColor==blue()).
```

There is a wide range of draw options, and they are accessed by the syntax indicated above. The "Clip" option as written turns clipping on, i.e., large values are shut off (the user can adjust the maximal value, if desired). Many more examples of this sort of thing are given in the book **[JS]**, and complete information is available through hyperdoc.

The user can graph parametric equations and surfaces through the use of the draw procedure as well. In fact graphs may be manipulated as objects in *AXIOM*. For a bit of whimsey, the built-in procedure "makeObject" was used to produce the picture given on this page. The *AXIOM* code is quite straightforward, and the first part of it is given here.

```
ruled(y1,y2,y3,g1,g2,g3) ==
  -- create expressions for the parameterization
  x : EXPR INT := y1 + s * g1
  y : EXPR INT := y2 + s * g2
  z : EXPR INT := y3 + s * g3
  -- return the three coordinates
  [x,y,z]

sphere(r,a,b,c) ==
  x : EXPR INT := r * cos(u) * cos(t) + a
  y : EXPR INT := r * cos(u) * sin(t) + b
  z : EXPR INT := r * sin(u) + c
  [x,y,z]

xx := ruled(t,t**2,1,cos(t),sin(t),t)

sp :=  makeObject(surface(xx.1,xx.2,xx.3),
                    t=-%pi/2..%pi/2,s=-2..2,_
                      var1Steps==35,var2Steps==35)

xx := ruled(t/2,t**2,1,cos(t)/2,sin(t)/2,t/4)

makeObject(surface(xx.1,xx.2,xx.3+1.2),
            t=-%pi..%pi,s=-2..2,space==sp,_
                      var1Steps==35,var2Steps==35)
```



The first call to makeObject creates the object sp, and the next one given above has the draw option space==sp which causes the graph argument to be added to the space sp. Following the lines indicated above, more scaling was done, more graphs were added to the space, and then spheres of various radii and locations were added. A more mathematical use of *AXIOM*'s graphical facilities can be found in **[LL]**.

Moving on to *AXIOM*'s object-oriented compiler, let me refer the reader to the 1989 article **[L1]**, where the basic concepts are discussed, and the book **[JS]**. Version 2.0 of *AXIOM* will provide a compiler for the A# programming language which has a syntax similiar to the current compiler for *AXIOM* but which generalizes many concepts and produces more efficient code. In addition, with A# in place, the user will be able to take advantage of interlanguage communications.

The compiler is used to produce the *AXIOM* library and can also be used by any user to produce new library files (or even replace system files). It would be impossible to list all of the mathematical expertise built into *AXIOM* in this space. The 742-page book [**JS**] is a good but brief introduction to what is present. To get an idea of the level of abstraction and extensibility possible, the reader might want to take advantage of the (p)reprint series at NAG, Inc. Send e-mail to Dr. Richard Luczak (`rl@nag.com`) for more information.

For an application of the full power of *AXIOM*'s compiler and the interactive mathematical environment, let me point to [**L2**] and [**L3**], where it was used to set up categories and domains of computation in order to derive formulas in a complex area of algebra, and [**AB**], where it was used to discover an unexpected theorem enabling the authors to give simpler proofs of results in [**A**]. (It is due to a large backlog that [**AB**] has appeared before [**A**]!) The reference [**L2**] also contains general information about the system.

Finally, release 2.0 of *AXIOM* will also have the "NAG-Link" in place. This is a facility which uses *AXIOM* and HyperDoc to link to the NAG FORTRAN library software over a network so that *AXIOM*'s environment can be used to manage accurate numerical calculations involving root finding, interpolation, optimization, integration, ODEs, PDEs, and statistical applications.

For general information contact John Zurawski at NAG, INC., 1400 Opus Place, Suite 200, Downers Grove, IL 60515 (`johnz@nag.com`).

For questions about *AXIOM* and technical support you may contact: Tom Ryan (`ryan@nag.com`) for the academic environment; Sheila Caswell (`caswell@nag.com`) or Tony Nilles (`nilles@nag.com`) for the industrial or government environments; and `axiom@watson.ibm.com` for technical support. Outside the Unites States contact `infodesk@nag.co.uk`.

References

[A] G.E. Andrews, *Plane partitions* V: *The TSSCPP conjecture*, J. Combin. Theory, Ser. A (to appear).

[AB] G.E. Andrews and W.H. Burge, *Determinant identities*, Pacific J. Math. **158** (1993), 1–14.

[C] L. Cardelli, *Basic polymorphic type checking*, Science of Computer Programming, vol. 8, (1987), pp. 147–172.

[JS] Richard D. Jenks and Robert S. Sutor, *AXIOM, The scientific computation system*, Springer-Verlag, New York (1992).

[L1] Larry Lambe, *Scratchpad* II *as a tool for mathematical research*, Computers and Mathematics column, Notices Amer. Math. Soc. (February 1989).

[L2] Larry Lambe, *Next generation computer algebra systems, AXIOM and the Scratchpad concept: Applications to research in algebra*, Plenary talk, 21st Nordic Congress of Mathematicians, Luleå, Sweden, summer 1992 (to appear).

[L3] Larry Lambe, *Resolutions that split off of the bar construction*, Journal of Pure & Applied Algebra, vol. 84, (1993), pp. 311–329.

[LL] Larry Lambe and Richard Luczak, *Object-oriented mathematical programming and symbolic and numeric interface*, 3rd International Conference on Expert Systems for Numerical Computing, May 1993 (to appear).

[MW] Thomas G. Moher and Paul R. Wilson, *Design of the opportunistic garbage collector*, Proc. 1989 Conference on Object Oriented Programming: Systems, Languages and Applications, ACM Press, New Orleans, LA, 1989, pp. 23–35.

Object Logo™ *Student Edition*

Reviewed by Suzanne M. Molnar*

Object Logo™ *Student Edition* is an implementation of the programming language *Logo* for the Macintosh. It is available from Paradigm Software Inc. (P.O. Box 2995, Cambridge, MA 02238; 617-576-7675) for $49.95. System requirements include a Macintosh Plus computer or greater with at least 1 megabyte of RAM (2 are recommended) and System 6.0.4 or later. The software is compatible with System 7 with 24-bit addressing. For the purpose of this review it was run on a Macintosh II with 5 megabytes of RAM and System 6.0.7.

Object Logo™ *Student Edition* provides the functionality of the mathematics and list processing of *Logo* and **turtle geometry**. In addition it supports an object-oriented programming environment. The full version of *Object Logo*™ ($195.00) includes a file compiler, application generator, MIDI (music) and robotics modules, and a complete 465-page *Object Logo*™ Reference Manual. At the time of this writing the full version was available for $135.00 for owners of the Student Edition.

The *Student Edition* comes with the 186-page book *Logo for the Macintosh: An Introduction through Object Logo*™ by Harold Abelson and Amanda Abelson [1]. After working through the first few chapters, the user has the groundwork for further exploration into turtle geometry, recursion, and list processing even if one has not programmed. This is a primary advantage if *Object Logo*™ is to be used by students with little or no programming background. If you have familiarity with the programming language LISP, from which *Logo*'s use of lists is adapted, the learning curve is a straight line with small slope!

There are three windows available to the user of *Object Logo*™, illustrated in Figures 1 and 2 on pages 19 and 20, respectively. When *Object Logo*™ begins, the Listener window appears with the ?-prompt. This is the window where interactive sessions occur. The Graphics window (or turtle window) also appears upon start-up, provided the *Object Logo*™ Elementary file—the program which controls turtles from the keyboard, mouse, and menu—is placed in the Startup Folder. One turtle appears at the center of this window. The third window is the file window for creating, editing, and saving programs.

Since *Object Logo*™ is interactive, procedures may be written in the Listener window without using the file window. The transcript of the Listener session can be saved but will not run, as it has responses interspersed with commands.

*Suzanne Molnar teaches mathematics and computer science at the College of St. Catherine, St. Paul, MN. e-mail: `smmolnar@alex.stkate.edu`.