# NOTICES
OF THE
AMERICAN MATHEMATICAL SOCIETY


## ARTICLES



## FEATURE COLUMNS



## DEPARTMENTS

But, if what you say is right what becomes of objectivity, an idea that is so definitively formalized by mathematical logic and by the theory of sets, on which you yourself have worked for many years of your youth?

There was visible emotion in his answer. Really? What makes you so sure that mathematical logic corresponds to the way we think? You are suffering from what the French call a *deformation professionelle.* Look at that bridge over there. It was built following logical principles. Suppose that a contradiction were to be found in a set theory. Do you honestly believe that the bridge might then fall down?

Do you then propose that we give up mathematical logic? said I , in fake amazement.

Quite the opposite. Logic formalizes only very few of the processes by which we actually think. The time has come to enrich formal logic by adding to it some other fundamental motions. What is it that you see when you see: You see an object *as* a key, you see a man in a car *as* a passenger, you see some sheets of paper *as* a book. it is the word "as" that must be mathematically formalized, on a par with the connectives "and," "or," "implies," and "not" that have already been accepted into a formal logic. Until you do that, you will not get very far with your AI-problem.

This sounds like an impossible task, said I, trying to sound alarmed.

It is not so bad. Recall another instance of a similar situation. Around the turn of the century, most mathematicians and physicists probably thought that the common-sense notion of simultaneity of events was an obvious one and needed no exact explanation. Yet a few years later Einstein came out with his special theory of relativity, which exactly analyzed it. I see no reason why we shouldn't do with "as" what Einstein did with simultaneity.

Do not lose your faith, concluded Stan. A mighty fortress is our mathematics. Mathematics will rise to the challenge, as it always has.

At this point, Stan Ulam started walking away, and it struck me that he was doing precisely what Descartes and Kant and Charles Saunders Peirce and Husserl and Wittgenstein had done before him in a similar juncture. He was changing the subject.

We never returned to this topic in our conversations, and since then, whenever I recall that day, I wonder whether or when AI will ever crash the barrier of meaning.

# Reviews

## Scratchpad II as a Tool for Mathematical Research

As is the case with many of you, I have had a bit of experience with some Symbolic Manipulation System. In my case, it has previously been with REDUCE, a system which I have used for research in Algebra [LS]. In the course of other research in Algebra and Topology [LL1], however, I had to develop special software using LISP since what was needed fell outside of what REDUCE handled. In this case, I needed to compute "exotic products" such as those arising in certain spectral sequence calculations [LL2]. One marked difference between my routines in REDUCE and those in LISP was in program readability: the LISP program was relatively incomprehensible to a colleague well versed in both REDUCE and LISP. This was not judged as a failing of the LISP language. Rather, REDUCE seemed to produce an environment where algebra packages end up better structured than if done in a general purpose environment such as provided by LISP. This alone would be a good reason for encouraging the development of mathematical packages in systems such as REDUCE. There are drawbacks however in both the use of REDUCE and LISP for such implementations. For example, it is not currently possible to create an algebra over some field and then use that algebra as coefficients of polynomials which are the entries of matrices used in some calculation in an *easy* and *convenient* way. Now let me make it clear that I am no ordinary user of computer algebra systems. Rather I am someone who has used and is continuing to use computer algebra for research in mathematics. For this reason, I ask that the reader please put my remarks in the correct perspective. There are many, many good features of REDUCE and other computer algebra systems such as user-interface matters which I am completely disregarding here. Similiarly, I am disregarding many features of Scratchpad II, the system I am reviewing from the mathematical research perspective. In particular, I am writing only about the *programming language* for Scratchpad II. This is the language used to create algebra facilities for the Scratchpad II library. The user

who wants to use the system for what already exists, (and that's quite a bit) uses the *interface language*. These two languages are very similiar except for type declarations. In the programming language, program constants, variables, and functions are required to have declared types (nothing is hidden). When using the interface language, however, type information can usually be omitted. Also, I will make some analogies with Pascal. This is only because that language is universally known and therefore a convenient one to use in order to communicate a point.

Like many other mathematicians, I don't remember where or how I first heard about Scratchpad. Perhaps it was through an old Scratchpad Newsletter forwarded to me by a colleague. I do remember that, at first glance, it seemed to be a system that I ought to have a good look at some day. I had an opportunity to do so this Summer when I visited the Computer Algebra Group in the Mathematics Department of IBM Research in Yorktown Heights, NY where the system is under development. At first sight I was delighted to see that Scratchpad had a familiar syntax which could be easily learned by anyone versed in Pascal. Having taught Pascal at several levels at the University of Illinois in Chicago, I have enough experience to say that the syntax should be just about as easy to learn as Pascal for someone not familiar with that language. A major point is that it also has all of the conveniences of the memory management available in a language like LISP. After spending about a week programming a few exercises, such as the solution of some easy partial differential equations by using complex congugate coordinate substitutions, multiple integration and inverse substitutions in the interpreter, I felt that it was time to create one of my own "domains." To explain what this means, let me give a very brief outline of the structures available in Scratchpad II.

Objects are organized by what are called "constructors." There are three main constructors available. They are: category constructors, domain constructors, and package constructors. These constructors may be thought of as being in analogy with Pascal's procedures, however their arguments may be other constructors of any type. Categories form hierarchies in a precise mathematical sense. For example, there is a field constructor and it is a subcategory of the ring constructor. The classification of structures is accomplished by the use of category constructors which produce conceptual classes. It was convenient for me to think of category constructors as containing the axioms for a structure which could be implemented in the system but possibly might not already be. Domain constructors require their arguments to be known objects and produce objects of some known class. It was convenient for me to think of domain constructors as models of the axioms given by some category constructor. Algorithms can be organized into package constructors which may be thought of as parameterized packages of functions—the parameters being other constructors. For example, one has the category constructors *Set*, *Ring*, and *Module*, and so, for example, the category *Module*($R$) can be formed where $R$ is a ring. One also has, for example, domain constructors *Integer*, *GaloisFieldF*, and *FreeModule*. The domains *GaloisField*($q$), and *FreeModule*($R, S$), where $q$ is a power of some prime, $R$ is a ring, and $S$ is a set, can therefore be formed.

All of the constructors (categories, domains, and packages) are described by programs written in the Scratchpad language itself. Any part of the system can be edited, modified, or extended by any user. Programs are compiled so that user extensions are as efficient as original code written by system implementors. It is important to emphasize that new structures can be built on what is already there.

One of the great advantages of Scratchpad comes from the fact that domains have been classified into categories. Because of this it is possible to write routines that operate uniformly over a category, and thus apply to any particular domain. Some other languages, e.g. Ada, have provided for a certain level of flexibility in this respect, but not, to the author's knowledge, to the extent present in Scratchpad. A consequence of this flexibility is the increased "reusability" of Scratchpad software. For example, if you needed to work with the exterior algebra on a finite set $X$ over a ring $R$, and it didn't exist, you could implement it by defining the correct monoid structure on a basis which could be extended linearly over any ring $R$. You would then be able to *immediately* use the exterior algebra over the rational numbers, or a representation of the real numbers, or over one of the finite fields $\mathbf{F}_{p^n}$. You could *immediately* produce an exterior algebra of differential forms by passing one of Scratchpad II's function rings to your domain constructor, and you could produce a package constructor implementing the exterior derivative of exterior differential forms in the domain just constructed. If you were inclined to investigate the cohomology of various Lie algebras, you could also produce a package constructor implementing the Lie algebra differential of Cartan-Chevalley-Eilenberg

in the exterior algebra on a dual basis of some Lie algebra.

I made such an implementation during my visit this Summer, and I'd like to present the ideas involved so that the reader may get an idea of how Scratchpad II allows one to "think mathematically" in programming domains which might be required in the course of some investigation. Here I will show fragments of the actual Scratchpad II code for these domains which is approximately 110 lines on three sheets of paper and is available from the author.

To begin, recall that by definition the exterior algebra $\Lambda X$ on a set $X$ is the free graded commutative algebra with multiplicative basis $X$ whose elements are to be considered as having degree 1. Of course other gradings are possible and easily implemented, but this will suffice for the example. A moment's reflection brings the following scheme to mind: represent the set $X$ by the set $\{1, 2, \ldots, n\}$. Then the set of monomials making up an additive basis for the exterior algebra is represented by the set $M$ of all ordered subsets of $X$. i.e. represent the basis elements $x_{i_1}, x_{i_2} \ldots x_{i_k}$ by the ordered subsets $(i_1, i_2, \ldots, i_k)$ of the set $\{1, \ldots, n\}$. The multiplication is then represented by the following operation on such ordered sets $A$ and $B$.

$$AB = \begin{cases} 0, & \text{if } A \cap B \neq \varnothing \\ (-1)^{\operatorname{sgn}(A,B)} \operatorname{sort}(A \cup B), & \text{if } A \cap B = \varnothing. \end{cases}$$

The nonnegative integer, $\operatorname{sgn}(A, B)$ is just the number of interchanges needed to sort the union of $A$ and $B$. Notice that our operation, as defined, is a function from $M \times M$ to $\mathbf{Z}(M)$ (the free Abelian group on $M$). This operation is to be extended to one which is defined on $R(M) \times R(M)$ and takes values in $R(M)$ (the free $R$-module on $X$).

I decided that two domains would be produced. One domain, OrderedExteriorAlgebraBasis (abbreviated OEAB), would represent the ordered basis mentioned above and the other, abbreviated ASYM (AntiSYMmetric), would take OEAB and a ring as a parameter and export all of the operations from the exterior algebra over that ring.

Before looking at the code a few general remarks are in order. Every procedure in the system must specify the type of its input parameters and return value. In order to do this, a "signature" for the procedure must be given. For example, to define the function $f(x) = x^2$, taking Integer to Integer, the following two lines of code suffice:

$$f : \textit{Integer} \text{ -> } \textit{Integer}$$
$$f(x) == x * x$$

the first line gives the signature of $f$ and the second line gives the implementation of $f$. If the procedure is $n$-ary, the domains are listed in the form of an $n$-tuple so that, for example, the operation of a group $G$ would have signature

$$*: (G, G) \text{ -> } G$$

Now for objects such as groups, there is usually a category constructor already in existence. The category constructor contains not only the signatures of the relevant operations, but also the properties that they may satisfy. The constructor can refer to previously defined category constructors for the definitions. If you are going to implement a group, you do not need to repeat the definition of the signature of the group operation since it is already contained in the category which you are modelling. In addition to the domain constructors already mentioned, there are also NNI (NonNegative Integers) and List($S$) (List of [elements of] $S$ where $S$ is a Set). Since $I$ is also a set, one can form List($I$) the domain which represents lists of integers. I must also explain that on Lists, Scratchpad has the built in functions first($A$) and rest($A$) which give the first element of $A$ and the list obtained by removing the first element of $A$. To refer to a domain under construction within the code for that domain, the implementor uses the symbol $. You will see a reference to "Rep" in the code below. Generally, the Rep is a type which will be built upon and, with the extra structure, becomes the new type exported. A forgetful function is provided in OEAB so that we may go from thinking of a monomial in the algebra to thinking of an ordered subset of $\{1, \ldots, n\}$.

Here's the domain OEAB: `OEAB`:

```
Exports == OrderedSet with
  degree : $ -> NonNegativeInteger
  forget : $ -> List(I)

Implementation ==
  Rep := List(I)      the representative type is List (I)
  x,y:$               x and y are of type OEAB and are
                      for local use. Equality in OEAB
                      is equality in the Rep Domain
  x = y == x =$Rep y

  x < y ==            "<" is the lexicographic order
    if first x = first y
      then rest x < rest y
      else first x < first y

  forget(x) ==
            x:Rep think of x as an element of List(I)
  degree(x) == #x   the number of elements of x
```

After compiling the domain OEAB, I had an Ordered Set constructor which I could use in my (as yet unwritten) Algebra constructor ASYM($R, X$) whose arguments are a ring $R$ and a set $X$. For ASYM, I will assume that a function "*exmerge*$(A, B)$" whose arguments are lists of integers and whose value is a list of integers already exists. The value of *exmerge*$(A, B)$ is a list which is empty if $A \cap B \neq \emptyset$ and whose first element is the sign sgn$(A, B)$ above and whose remaining elements consist of the sorted union of $A$ and $B$ otherwise. I used the expression *Nul* to denote the empty set $\emptyset$. In the version of *FreeModule*$(R, M)$ that I used, the following operations are exported: for an element $x$ of *FreeModule*$(R, M)$ there is *destruct*$(x)$ which is a list of two element lists $t$ such that the first element of $t$ is an element of $M$ and the second element of $t$ is an element of $R$. The idea here is that the element $r_1 m_1 + \cdots + r_k m_k$ is represented by the list $[[m_1, r_1], \ldots, [m_k, r_k]]$. The zero element of the module is represented by $[[Nul, 0]]$. The first element of a term $t$ is referenced by *t.mon* (the monomial part) and the second element by *t.coef* (the coefficient of this term). Let *Term* denote the type of such $t$'s. There also is the function *construct*$(l)$ which is precisely the inverse of *destruct*. I chose to make the set $X$ which gives the multiplicative basis for $\Lambda(X)$, and the ring $R$ which gives the coefficients, the arguments of the constructor. I should also point out that the operations *LeadingCoef*, *LeadingTerm*, and *reductum* are present in the domain *FreeModule* and have the obvious meaning except for *reductum* which is a function that takes a sum of terms and returns the sum minus the leading term. These signatures are given in the code for ASYM without any implementation—the effect is that these operations will be exported from the domain ASYM *as they are* from *FreeModule*. Finally, the construction $+/L$ is familiar to anyone who has seen APL. It returns the result of summing all of the elements from the list $L$. The end result of all of this is a domain ASYM$(R, X)$ which may be used as follows (in the interpreter, if you like): the assignments $X :=$ $[dx, dy, dz, dt]$ and ext := ASYM$(EF \quad RN, X)$ create the exterior algebra on a vector space of dimension 4 with multiplicative basis $\{dx, dy, dz, dt\}$ over the ring of elementary functions with rational number coefficients. Working with the elements of *ext* is then as easy as working with ordinary polynomials in the system.

Currently, there are around 600 constructors in the system which requires 12 MB of real memory and about 180 MB of disk space running on an IBM RT/PC. A nice user interface and graphics are currently being added. Scratchpad II is not yet commercially available, but several Universities have received copies of the system through a joint research agreement with IBM.

In summary, Scratchpad II is both a computer algebra system and a serious programming language—the compiler is being written in this language; its design allows the user to mix the two ideas to his advantage. A great degree of software reusability is achieved by organizing domains into categories. Anyone who is thinking about doing mathematical programming in any area theoretical or applied should take a good look at this system.


[LL1] L. Lambe, *Cohomology of principal G-bundles over a torus when $H^*(BG; R)$ is polynomial*, Bull. Soc. Math. de Belg. **38** (1986), 247-264.

[LL2] L. Lambe, *Algorithms for the homology of nilpotent groups*, Conf. on Applications of Computers to Geom. and Top., Dekker Inc., N.Y. (to appear).

[LS] L. Lambe and B. Srinivasan, *A computation of Green functions for some classical groups* (preprint).

[TH] A. Hearn, *REDUCE User's manual*, The Rand Corporation, Santa Monica, Ca 90406-2138, July, 1987.


Here is the main body of the exterior algebra domain. In the code for "*," | should be read as "such that" and ^ as "not."

```
ASYM(R,X):

Exports == Algebra(R) with

  LeadingCoef      :$          -> R
  LeadingTerm      :$          -> $
  reductum         :$          -> $
  coef             :($,OEAB) -> R

Implementation == FreeModule(R,OEAB) add

  Rep := FreeModule(R,OEAB)

  x,y:OEAB
  a,b:$
  r:R
```

```
coef(a, x) ==
    for t in destruct(a) repeat
        if t.mon = x then return t.coef
        if t.mon < x then return 0
    0

alpha: (R,OEAB,OEAB) -> Term
alpha(r,x,y) ==
   r = 0                          => [Nul,0]
   x = Nul                        => [y,r]
   y = Nul                        => [x,r]
   z := exmerge(forget x,forget y)
   z = Nul                        => [Nul,0]
   first(z) = 1                   => [rest(z),r]
   [rest(z),-r]

a * b ==
  +/[construct([u for tb in destruct b |
    ((u:=alpha(ta.coef * tb.coef,
    ta.mon, tb.mon)).mon) ^= Nul])
    for ta in destruct a]
```

Larry A. Lambe
University of North Carolina
Chapel Hill, NC 27514
lal@uncmath.math.unc.edu

## Graphical Aids for Stochastic Processes

"*Graphical Aids for Stochastic Processes*," Bob Fisch, David Griffeath. Belmont, CA: Wadsworth & Brooks/Cole, 1988. Contents: 6 diskettes for IBM compatibles with CGA or EGA graphics; manual, 11 pages. Cost $99.

The term "software" originally referred to compilers, interpreters, and operating systems—programs used to create other programs. As computers evolved, software came to include programs for creating files and working with data. This included word processors, spreadsheets, data analysis programs, and communication programs. Today software refers to almost any computer program with design features making it convenient to use, and giving it some chance of life beyond one particular project.

A type of software that has become prominent in the field of mathematics education is the computer video. Computer videos serve the same purpose as films and videotapes. They are to be watched. They provide demonstrations and tutorials. They do not usually have features for communicating with other software. They are rarely used to produce disk files. Computer videos are simple to use, though they have varying degrees of interaction with the user. Some only let the user determine when to move on to the next scene, while others, in the tutorial style, pose questions and evaluate answers given by the user. Some include educational games. Computer videos may be evaluated on the basis of seven criteria: Content, ease of use, user interaction, graphics, sound, text, documentation.

The software under review here is a set of six computer videos on probability and stochastic processes. The subjects of the videos are: Bernoulli trials, random walks, Poisson processes, Markov chains, branching & queueing, and Brownian motion. Each of the videos is subdivided into eight sections and takes about 50 minutes to view, if the viewer moves right along. Most sections have simulations. A few contain interesting educational games.

The first three videos are relatively elementary. The simulations include balls falling through an array of pegs, a drunken toad on a desert highway, and a fisherman whose name is Poisson. These are engaging simulations. They are interactive to the extent that the user may modify some parameters. They touch on the law of large numbers and the central limit theorem. They also touch on the three great distributions of applied probability theory: the binomial, the normal, and the Poisson. Anyone who likes probability theory will enjoy these videos.

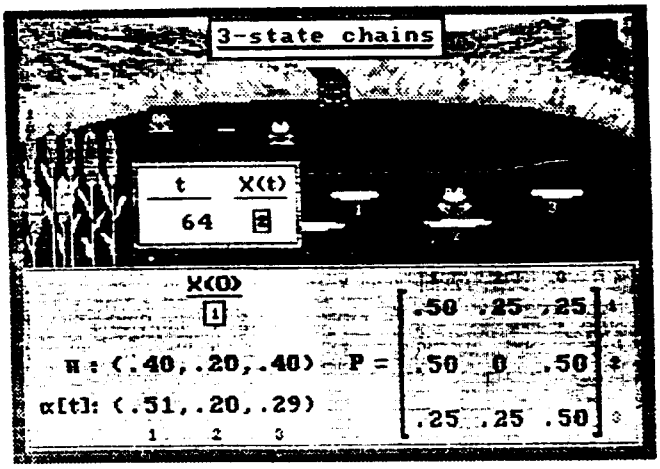


Figure 1. GASP screen image for Markov chains

The Markov chain simulations include a frog hopping around on three lily pads (Figure 1) and bugs walking along the edges of a graph. The sections on branching touch on martingales and contain a simulation for Galton's classic problem on survival