Astérisque

MARTIN C. TANGORA

Computing with the lambda algebra

Astérisque, tome 192 (1990), p. 79-89

<http://www.numdam.org/item?id=AST_1990__192__79_0>

COMPUTING WITH THE LAMBDA ALGEBRA

Martin C. Tangora

University of Illinois at Chicago
and University of Oxford

We are trying to compute the homotopy groups of spheres. This is an old problem now, and a very deep one; and a lecture on the subject is likely to be very technical. A number of the experts who know these technicalities are among the participants in this Congress. If I were to give a technical talk on my work, a few of you would already know what I was going to say before I had begun, but most of you would still not know what I had said after I had finished. Moreover, I have published elsewhere [6] a detailed account of several aspects of the problem that I am currently engaged in.

Accordingly, I would like to confine myself here to some remarks that I hope will be appreciated by everybody. On the one hand, I would like to give an idea of how we have managed to convert an effectively computable but realistically intractable problem into a tractable and really computable one. Here I will oversimplify the description, since the interested reader can refer to more detailed versions in the literature.

On the other hand, I would like to share with you some reflections on the meaning of "proof" as it is variously used in our various disciplines. When is a proof really a proof? Let me begin with an assertion made some years ago by the American humorist Al Capp, or rather by a character in his comic strip Li'l Abner.

Mammy Yokum's Principle : Good is better than evil, because it's nicer.

Mammy Yokum's method of proof is well known in many other fields of human endeavor, but I submit that it has been neglected by mathematicians and computer scientists

S.M.F.
Astérisque 192 (1990)

There is a saying among mathematicians that only a graduate student really knows what a proof is. Perhaps it is necessary to have published an erroneous result in order to appreciate this.

When we think we have a proof, we submit it to three tests. First, we try to find a mistake in it. Second, we submit it for publication, and the referee tries to find a mistake in it. Third, it is published, and everyone else tries to find a mistake in it.

This leads me to what I think of as the Mammy Yokum Test for a proof in mathematics : a proof of a mathematical result is a good proof if nobody has found a mistake in it.

Later we will offer a Mammy Yokum Test for the correctness of a computation.

1. ALGEBRAIC APPROACHES TO THE HOMOTOPY PROBLEM.

Homotopy groups are algebraic objects occurring in, and defined in, a purely topological setting. The definition is a matter of topological spaces and continuous functions. However, a dozen years after the definition had been codified, the great difficulty of the problem of computing these groups had become apparent; Hopf complained around 1950 that almost every known result had been obtained by a different method [4].

Great progress was made in the 1950s and 1960s, but the improvement came at the expense of elaborate techniques, and the result was a bewildering confusion of data, in which a variety of patterns can be seen to interact in complicated and often mysterious ways. It is known, for example, that every possible positive integer occurs as the order of an element in the homotopy groups of spheres. I attach a little table of the homotopy groups of the 6-sphere, extracted from Toda's 1962 book [7], and invite you to try to extrapolate to the next few groups.

The table gives, for each n, the order $O(n)$ of the homotopy group $\pi_n(S^6)$. A zero denotes a trivial group, and ∞ denotes an infinite cyclic group.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O(n)	0	0	0	0	0	∞	2	2	24	0	∞	2	60	48	8

n	16	17	18	19	20	21	22	23	24	25
O(n)	144	2016	240	6	24	360	2016	16	288	8448

More optimistically, Brown proved in 1957 that the homotopy groups of any finite complex are effectively computable [2]. However, Brown himself emphasized that his algorithms were not intended to be of any practical use.

Meanwhile, work of Steenrod, Cartan, Serre, and Adams led to the consideration of certain algebraic approximations to the homotopy problem, which are more amenable to computation. In particular the Adams spectral sequence converges to a filtered version of homotopy groups, and its E_2 term is an algebraic object for which any finite range can be obtained from a variety of algorithmic processes. The issue becomes one of efficiency : all algorithms are effective, but some are more effective than others.

We will confine ourselves now to the problem of computing the E_2 term of the Adams spectral sequence for spheres, for $p = 2$. At least four different methods have been used. Since the E_2 is the cohomology of the Steenrod algebra, it can be obtained as the homology of the cobar construction; but this construction is very large and the method is too slow and cumbersome. Adams used a minimal resolution, but this method seems awkward for large computations, although recently Bruner has had surprising success with it on a computer. The May spectral sequence is well suited to hand computation but to date has not been carried any further on a machine.

We focus on a fourth method, which obtains the Adams E_2 term as the homology of the lambda algebra. When this algebra was announced in the 1960s, it did not seem very promising for computation, but Ed Curtis showed the way. George Whitehead has done some extensive calculations by hand, and the ideas of Curtis are amenable to algorithmic development and to machine computation.

2. THE LAMBDA ALGEBRA.

For each prime p there is a lambda algebra. To simplify the discussion we will only consider $p = 2$. In this case Λ is an associative bi-graded differential algebra over the field of 2 elements, with a generator λ_n in each non-negative dimension n. The algebra is *not* commutative, so your monomials are ordered products λ_I where $I = (i_1, i_2, \ldots, i_s)$ is a sequence of non-negative integers. It is natural to write I as an abbreviated notation for λ_I. The algebra is defined by the relations

$$\lambda_i \lambda_{2i+n+1} = \sum_{j \geq 0} A(n,j) \lambda_{i+n-j} \lambda_{2i+1+j} \qquad (i \geq 0, n \geq 0)$$

and the differential

$$d(\lambda_{n-1}) = \sum_{j \geq 1} A(n,j) \lambda_{n-j-1} \lambda_{j-1} \qquad (n \geq 1)$$

where $A(n,j)$ denotes the binomial coefficient $\binom{n-j-1}{j}$ reduced mod 2. The bi-grading of a monomial indexed by I may be written (r,s) where s, as above, is the length of I, and $r = i_1 + \ldots + i_s$. Using the relations we can express all monomials in terms of the "admissible" ones satisfying $2i_j \geq i_{j+1}$ $(1 \leq j \leq s-1)$.

Because of the non-commutativity, the algebra grows very fast. Just by counting the elements (using a computer, of course) we find an exponential growth rate of 1.79 with respect to the r grading. This was recently explained by Flajolet and Prodinger [3]. Since 1.79^{10} is about 345, we see that if you have an algorithm that is linear with respect to the number of elements in the admissible basis, and if you can compute E_2 from dimension $r = 30$ to dimension $r = 40$ in a month, then you can go from 40 to 50 in about thirty years. This may be effective, but it is not effective enough.

As anyone knows who has worked in computational linear algebra, the key is to choose the right basis. Curtis's method of choosing bases may have been motivated topologically, but it has the interesting effect of allowing us to set aside the vast majority of monomials as being irrelevant. The following discussion is intended to give a rough idea of what I mean by this, and to identify the properties of the lambda algebra that make this possible.

Using the natural ordering of the generators, we can proceed to order monomials, polynomials, and even sets of polynomials. Thus each element of a certain bi-grading is put in "proper form", meaning that not only are the relations used to express everything in terms of the admissible basis, but that then the resulting polynomial is written with its largest term first. We then seek to determine the minimal representative of each homology class, the minimal basis for cycles, and for each minimal cycle that bounds, the minimal element among those that it bounds, called its "tag". The output of our computation is what I call a "Curtis table". In such a table one finds entries of the form c/t, where c and t are monomials; c is the leading (maximal) term of a cycle in the minimal basis for the cycles; and t is the leading term of the smallest element that has c as the leading term of its boundary. It might seem more natural to give the complete polynomials of which c and t are the leading terms, but it is crucial to the success of our technique that it is usually not necessary to keep track of the other terms.

Why not ?

Our method is based on the following relations between the differential and the ordering.

(1) In the differential $d(\lambda_i)$, all terms begin with λ_n where n is less than i. (Immediate from the definition.)

(2) In the product $d(\lambda_i)\lambda_j\lambda_k\ldots$, all terms begin with λ_n where n is less than i. (Proved by Wang [9].)

(3) If $\lambda_n z + y$ is a cycle in proper form, then z itself is already a cycle. (Follows easily from (2).)

From these relations it is easy to deduce the following key properties of the "tags" in the Curtis table.

(4) Cancellation : If $\lambda_i x$ is the tag of $\lambda_i z$, then x is the tag of z.

(5) Propagation : If x tags z then $\lambda_i x$ tags $\lambda_i z$ (provided these are both admissible).

Roughly speaking, if the cycle z is the boundary of y, then multiples of z are boundaries of the corresponding multiples of y. Thus most of the tags in

the table follow immediately from earlier tags. Each entry z/y gives rise to a multitude of entries sz/sy, where s is any string of generators such that sz and sy are admissible. We not only don't need to compute these, we don't even need to record them, since they can always be recovered by cancelling the initial string s.

Here is an elementary analogy. When we learn to multiply the natural numbers, we memorize the multiplication tables for multiples of $2, 3, \ldots, 9$. There is no need to "store" the multiples of 0, or of 1, or of 10. At the age of eight, none of us was brilliant enough to invent the binary system; but if we write our numbers in the base 2, all our multiplications are with 0, or 1, or 10, and there is nothing to store. Properties (4) and (5) are like division and multiplication by 10, but they are valid for *every* generator.

Of course, in elementary arithmetic, changing to base 2 does not eliminate all the work; it shifts the work from multiplication to addition. In our lambda-algebra algorithms, the suppressed computations must often be called up in order to complete the later work. Still, the net benefit is substantial.

For example, in the bi-grading $(39, 17)$ there are more than one hundred million admissible monomials, but the fate of all but three is determined inductively by the cancellation property. One, with $I = (6, 1, \ldots)$, is easily seen to be a boundary. Another, with $I = (2, 4, \ldots)$, is easily seen to be a cycle. Of the six non-obvious elements at $(40, 16)$, one begins with a 2 and thus is too small to tag $(2, 4, \ldots)$ (by property (4)); one is used to tag $(6, 1, \ldots)$; and the other four are easily seen to be cycles; so $(2, 4, \ldots)$ represents a homology class. All this is done quickly by inspection. However, the remaining entry at $(39, 17)$ is not so easily settled. It took 25 minutes of CPU time on an IBM 3081D to show that this entry is a cycle (and therefore represents another homology class).

Another measure, admittedly imprecise, of the success of this approach is the following. The growth rate of the lambda algebra itself is exponential, and can be proved, either by actual counting or by the method of Flajolet and Prodinger, to obey the asymptotic relation

$$n(t) \sim (.283)(1.48)^t$$

where t is the *total degree* ($= r+s$ in the notation above) and $n(t)$ is the number of basis elements in that degree. But if we plot the CPU times for each t against t, we find a significantly smaller growth rate:

$$CPU(t) \sim k(1.32)^t$$

where the constant k is .00478 seconds for the mainframe used when degrees $t = 27$ to 48 were computed in 1981 (see 4.8 of the Memoir [6]).

What is noteworthy is that the growth rate of the computing time is lower than that of the algebra, despite the (sometimes staggering) increase in the number of terms in a cycle, the number of factors in each term, the number of terms in the differential of each factor, etc.

3. IMPLEMENTATION

How do you put this problem onto a machine? A list-processing language is a natural medium, since polynomials are naturally coded as linked lists. We chose to write our programs in SNOBOL4, a general-purpose language with good list-processing facilities and quite suitable for symbolic algebra.

When I was developing these programs I was teaching a course in list-processing languages that give students an introduction first to LISP and then to SNOBOL. So an interaction took place between teaching and research! The problem of adding two polynomials mod 2 is a good exercise in such a course: given two sorted linked lists, merge them into a single sorted linked list, eliminating identical terms two by two.

An issue that arises is whether this SUM procedure must leave its arguments intact. If you need to conserve storage, you would like to build your SUM without creating any new nodes, just by manipulating the links (pointers) between the nodes in the two arguments; but this destroys the arguments.

The SNOBOL4 language has the reputation of being large and slow. At the University of Illinois at Chicago, however, we had access to a SPITBOL compiler on an IBM mainframe. Eventually we were using up to four hours of CPU time and up to five megabytes of storage. The ultimate limitation

on our computations was neither time nor storage, but a consequence of the intense list-processing activity: when the program was competing for the CPU in a multiple virtual storage environment, it was brought to a standstill by excessive paging activity.

In trying to optimize the programs with respect to time and storage constraints it was necessary to make some compromises. In the inductive lambda-algebra computations you often enter a loop and call for the differential of a certain monomial several times. Therefore, at one extreme, you could have your program store each differential, using the extra storage to save the time of re-computing. At the opposite extreme you could re-compute every binomial coefficient whenever a procedure called for it. In the actual work we store binomial coefficients, the defining relations, and the differential on the generators. The difficult question is whether to store the differential on the monomials. For a long time we did this locally at each bi-grading.

Notice how this impinges on the question of whether the SUM procedure must preserve its arguments. If one summand is taken from a table, and the SUM procedure changes the pointers around, you lose the integrity of your table.

This brings us back now to the question of proof. We alluded briefly above to loops that arise in the inductive computation. The ordering and the differential have an interesting and useful relationship, as shown by properties (1)-(5), but nothing is perfect. Neither the function $c = d(y)$ nor the relation "x tags z" is monotone. Consequently there is a difficulty in proving not only that the algorithm is correct, but even in proving that it is finite.

It took me a little time to see how these proofs should go. As a mathematician I wanted to prove that the results were correct. This seems quite difficult. Eventually I came to understand that the right idea is to prove that the *process* is correct.

I had read the discussion in Knuth's book about proving an algorithm correct [5]. In principle you represent your algorithm by a flowchart or its equivalent, and label every arrow with a certain set of assertions, so that the validity of

all these assertions is clear when you begin the process and remains in force at every step of the process.

Looking through the literature, I did not find very many proofs written down in such fashion. In mathematics, we like to see the details, but I got the impression that detailed proofs of correctness of algorithms do not often get into print. I ended up presenting my proofs of the correctness and finiteness of the lambda-algebra algorithms in informal and summary fashion. So far they have satisfied the three tests; at least I, the referee, and Curtis have not found any mistakes. However, I recently found a gap.

4. GETTING THE RIGHT ANSWER.

One of the advantages of the lambda-algebra over other fast methods of computing the E_2 term of the Adams spectral sequence is that all the product structure of E_2 is present. The cohomology of the Steenrod algebra does not have just ordinary products, but comes endowed with a rich higher product structure, including both Steenrod reduced powers and Massey products. The more you know about these higher products, the more you can deduce about the Adams spectral sequence and homotopy groups.

We won't stop to explain what a Massey product is here, but when three or more homology classes satisfy certain relations, you can form a Massey product on them [8]. Usually it is not uniquely determined, but comes with a certain "indeterminacy", which is our way of saying that it's actually not an element of the homology group but rather a coset. Anyway, to compute such a thing you only need to be able to multiply in the ordinary sense, and when an ordinary product is a boundary you need to find something that it's the boundary of. What we call a "tag" is not only such a thing but it's a canonical such thing, because it's minimal. So the operations used to construct a Massey product are more or less the same operations that have already been programmed in building the Curtis table.

So I went back to the programs that I had proved correct, took some of the procedures, saluted the flag marked Structured Programming, and started

computing Massey products. Before long I found some serious errors in the results from the new programs. How could this be?

The difficulty turned out to be related to the list-processing issue that we discussed above. In the Massey-product programs, the SUM procedure was being used on arguments that were needed again later. Since the SUM procedure altered its arguments, serious errors resulted.

It was easy to write a new procedure, called SAFESUM, that left its arguments inviolate. But what did all this imply about my proofs of correctness?

One of the hidden assumptions in my proof had been that if a differential was correctly entered into a table, then it remained correct in the table. This assumption was never made explicit, of course, and it never caused trouble in the earlier programs. After the mistake was found in the new programs, it was easy to go back to the earlier ones and verify that in those programs the SUM procedure was never called on any argument that would be used again afterward. Actually the logic of the algorithms is very tight, and after many hours of CPU time without any errors coming to light, one is virtually certain that no such mistake has been made. Still, one can say that a gap in the proof has been closed.

Notice the light that this sheds on the difference between the correctness of an algorithm and the correctness of a program.

We conclude, with Mammy Yokum, that a computation is correct if it gives the right answer.

REFERENCES.

[1] A. K. Bousfield, E. B. Curtis, D. M. Kan, D. G. Quillen, D. L. Rector and J. W. Schlesinger. *The mod-p lower central series and the Adams spectral sequence.* Topology, 1966, vol. 5, pp 331-342.

[2] E. H. Brown Jr. *Finite computability of Postnikov complexes.* Ann. Math., 1957, vol. 65, pp 1-20.

[3] P. Flajolet and H. Prodinger. *Level number sequences for trees.* Discrete Math., 1987, vol. 65, pp 149-156.

[4] H. Hopf. *Vom Bolzanoschen Nullstellensatz zur algebraischen Homotopietheorie der Sphären.* Jahresbericht der deutschen Mathematiker Vereinigung 56 (1953) 59-76.

[5] D. E. Knuth. *The art of computer programming.* 1973, vol. 1, 2nd ed. (see Section 1.2.1, pp 14-16).

[6] M. C. Tangora. *Computing the homology of the lambda algebra.* Memoirs Amer. Math. Soc. No. 337 (1985).

[7] H. Toda. *Composition methods in homotopy groups of spheres.* Princeton, 1962.

[8] H. Uehara and W. S. Massey. The Jacobi identity for Whitehead products. In *Algebraic geometry and topology, a Symposium in Honor of S. Lefschetz,* Princeton University Press, 1957, pp. 361-377.

[9] J. S. P. Wang. *On the cohomology of the mod-2 Steenrod algebra and the non-existence of elements of Hopf invariant one.* Illinois J. Math., 1967, vol. 11, pp 480-490.

Author's address:
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
Chicago, Illinois 60680 U.S.A.

The above talk was presented on August 31, 1987, at the International Conference on Computational Geometry and Topology and Computation in Teaching Mathematics, Universidad de Sevilla.