Astérisque

KATSUYUKI SHIBATA

Micro-computer prolog as a handy tool for formal algebraic computations

Astérisque, tome 192 (1990), p. 69-78

<http://www.numdam.org/item?id=AST_1990__192__69_0>

MICRO-COMPUTER PROLOG AS A HANDY TOOL FOR FORMAL ALGEBRAIC COMPUTATIONS

Katsuyuki Shibata

Prolog is a logic programming language, and its grammer is based on the first order predicate logic. It has in itself an inference mechanism which runs automatically. Since logic and mathematics are near in a naive sense, it is not surprising that sometimes the translation from mathematical formulas to a Prolog program is straightforward and that they look very similar. I will shortly show it by explicit examples.

From this point of view, Prolog seems to be a very good language for those mathematicians who are not specialists of computer sciences and who are not so much interested in learning the details of computer mechanisms.

But let me first explain about the Gelfand-Fuks cohomology of a smooth manifold. My experience on micro-computer Prolog was to compute a part of that cohomology in the sphere case.

§ 1. Gelfand-Fuks cohomology

Let M be a paracompact Hausdorff smooth manifold and $\mathcal{L}_M$ be the Lie algebra of smooth tangent vector fields on M equipped with C^∞-topology. ($\mathcal{L}_M$ is also denoted as $\mathfrak{X}(M)$ or as Vect(M).) And let $C^*_c(\mathcal{L}_M) = \bigoplus_{q>0} C^q_c(\mathcal{L}_M)$ be the Koszul complex of continuous cochains of the topological Lie algebra $\mathcal{L}_M$. Namely the graded vector space $C^q_c(\mathcal{L}_M)$ is defined to be the set of all the continuous, alternating q-linear forms

S.M.F.
Astérisque 192 (1990)

$$f : \mathcal{L}_M \times \cdots \times \mathcal{L}_M \longrightarrow \mathbf{R} \ .$$
$$(q \text{ times})$$

And the differential $d : C^q_c(\mathcal{L}_M) \longrightarrow C^{q+1}_c(\mathcal{L}_M)$ is defined by the formula

$$(df)(X_1, X_2, \ldots X_{q+1}) = \sum_{1 \le i < j \le q+1} (-1)^{i+j} f([X_i, X_j], X_1, \ldots \overset{i}{\overset{\vee}{\ }} \ldots \overset{j}{\overset{\vee}{\ }} \ldots, X_{q+1}).$$

By the Jacobi identity of Lie algebra, $d \circ d \equiv 0$ holds and we can take the cohomology of $C^*_c(\mathcal{L}_M)$ with respect to d.

DEFINITION. The Gelfand-Fuks cohomology $H^*_c(\mathcal{L}_M)$ of the manifold M is defined to be the cohomology $H^*(C^*_c(\mathcal{L}_M); d)$.

The Gelfand-Fuks cohomology is related to the theory of exotic characteristic classes of foliations and is interesting in various aspects.

Gelfand and Fuks proved, among other things, the following finiteness theorem for the additive structure of $H^*_c(\mathcal{L}_M)$.

THEOREM (Gelfand-Fuks [1]).

If $\dim_{\mathbf{R}}(H^*(M; \mathbf{R})) < \infty$, then $\dim_{\mathbf{R}}(H^q_c(\mathcal{L}_M)) < \infty$ for every q.

In contrast to this, we have proved the following theorem concerning the multiplicative structure of $H^*_c(\mathcal{L}_M)$.

THEOREM (Shibata [4]).

If $\dim_{\mathbf{R}}(H^*(M; \mathbf{R})) < \infty$, then $H^*_c(\mathcal{L}_M)$ is finitely generated as an $\mathbf{R}$-algebra if and only if either of the following two conditions holds;

(1) $\dim M \leqq 1$ (ie. M = a finite union of {pt}, S^1, and $\mathbf{R}^1$), or

(2) $\bar{H}^*(M; \mathbf{Q}) \equiv 0$ (ie. M is rationally acyclic).

For the proof of this theorem, we computed Haefliger's model for $C^*_c($

$\mathcal{L}_M$) constructed by using Sullivan's minimal model theory in rational homotopy theory. This computation was done by hand, but later we became interested in using a computer for computations in explicit examples.

In case M is the n-dimensional sphere S^n, Haefliger's model reduces to the Koszul cochain complex $C^*(H^*(S^n;\mathbf{R})\otimes L(V_n))$ of Lie algebra $H^*(S^n;\mathbf{R})\otimes L(V_n)$, where V_n is a certain finite dimensional graded vector space depending on n, $L(V_n)$ is the free graded Lie algebra over V_n, and the Lie product in the above tensor product is defined as

$$[x\otimes\ell, x'\otimes\ell'] = \pm xx'\otimes[\ell,\ell'].$$

The ordinary (not necessarily continuous but all cochains) cohomology of this Lie algebra is isomorphic to $H^*_c(\mathcal{L}_{S^n})$ (Haefliger [2]).

We now know that this cohomology algebra is not finitely generated if $n \geq 2$, but our knowlege is far from complete. There are too many multiplicative generators. Therefore we are interested in computing the cohomology of the Lie algebra $H^*(S^n;\mathbf{R})\otimes L(V_n)$.

To begin with, we must know in detail the product structure of a free Lie algebra. To avoid tedeous sign calculations, I neglect the odd degree elements of Lie algebra in the explanations of the following sections.

§ 2. Hall basis criteria program

Let V be a vector space (over $\mathbf{Q}$ or $\mathbf{R}$) and $B = \{x_1, x_2, \ldots\}$ be a well-ordered basis of V.

DEFINITION. A well-ordered subset $H \subset L(V)$ is a Hall set relative to B if

(H-1). $H = \bigcup_{n\geq 1} H_n$, where H_n consists of elements of length n, $H_1 = B$, and the ordering in H satisfies the condition

$x < y$ if (length x) < (length y),

(H-2). $H_2 = \{[x,y];\ x,y \in B,\ x < y\}$, and

(H-3). $\bigcup_{n \geq 3} H_n = \{[Y,[X,Z]];\ X,Y,Z,[X,Z] \in H,\ Y \geqq X,\ Y < [X,Y]\}$.

It is known that a Hall set is an additive basis for L(V).

Given a Lie product element in L(V), we want to know whether it belongs to H or not. I am going to write down a Prolog program for that.

For simplicity's sake, I treat only the case where dim V = 2. The case for dim V = n > 2 is completely analogous. So let us assume B = {x,y} with x < y.

```
          /* Hall basis criteria program */
hall_basis(x).
hall_basis(y).
hall_basis([x,y]).
hall_basis([Y,[X,Z]]):-
     hall_basis(X), hall_basis(Y), hall_basis(Z), hall_basis([X,Z]),
     (Y=X ; smaller(X,Y)), smaller(Y,[X,Z]).
smaller(x,y).
smaller(X,Y):- lie_length(X,M), lie_length(Y,N),
     (M<N ;
       (M=N, X=[U,V], Y=[W,Z], (smaller(V,Z) ; (V=Z, smaller(U,W))))).
lie_length(x,1).
lie_length(y,1).
lie_length([X,Y],N):- lie_length(X,L), lie_length(Y,M), N is L+M.
```

In Prolog, each logical line ends with a full stop. A logical line may be written in several phisical lines if it is long. Each logical line is called a Horn clause. (Horn is the name of a logician.) There are two kinds of Horn clauses; those containing the symbol ":-" and those without it. The symbol ":-" means the logical "if", and "A:-B" means "statement A is true if statement B is true." This type of Horn

clauses are called rules, while the ones without ":-" are called facts because they are supposed to be true without any condition.

A statement is expressed by a predicate and its arguments. The predicate precedes the round brackets and the arguments come between the round brackets. You may use any combination of letters and symbols beginning with a lower-case letter as a name of predicates or arguments. A sequence of letters and symbols beginning with an upper-case letter is a variable. A Prolog variable can take as its value any combination of numbers, letters and symbols. It is because of this complete arbitrariness of names and values that we can employ in Prolog the similar formulas and terminologies as in mathematics.

Now that the above Hall basis program is put into your computer's main memory, let's start the execution. First you will see on the display screen the prompt symbol "?-". Prolog system prompts you to ask him a question. If you type in, after the prompt "?-", a question "hall_basis([x,[x,[x,[x,[[x,y],[x,[x,y]]]]]]]).", you will immediately obtain the answer "no". If you ask "?- hall_basis([[x,y],[x,[x,[x,[x,y]]]]]])." then Prolog responds "yes".

As soon as a question is typed in, Prolog system begins to look up in the program in memory to find a fact or a rule whose pattern matches that of the posed question. If it finds a suitable rule, it then tries to verify the satisfying conditions of that rule (ie. the statements following the "if" symbol) by the same pattern matching procedure. This continues recursively untill when all the statements are verified to be true by matching some facts in the program. This process can be interpreted as the execution of Robinson's resolution principle in the axiomatic proof theory.

Prolog's inference mechanism explained in the preceeding paragraph is too primitive to obtain from our Hall basis criteria program the list of all the Hall basis elements up to a certain length. But it is not difficult to modify that program, adding some executional strategical data, to obtain such a list of basis elements.

§ 3. Further free Lie algebra calculation program

By the very definition of an additive basis, any element in a free Lie algebra can be uniquely expressed as a linear combination of the Hall basis elements. We call this transformation of an element X into such a linear combination C the normalization of the element X, and express it as a Prolog clause "normalize(X,C)".

This normalization procedure is realized by a remarkably simple Prolog program. It is sufficient to put the following rules from ① to ⑪ in the program. Most of them are simply expressing the linearity and bilinearity. A proof that these rules suffice to normalize any given element was given in Shibata [5], and the outline of the proof is reviewed in Shibata [6]. It is a rather tedious proof and we used triple induction arguments.

Now the normalization rules.

```
normalize(X+Y,C):- normalize(X,A), normalize(Y,B), C=A+B    ...①
normalize(X-Y,C):- normalize(X,A), normalize(Y,B), C=A-B.   ...①
normalize(N*X,C):- normalize(X,D), C=N*D.                   ....②
normalize(-X,C):- normalize(X,D), C=-D.                     .....②
normalize([X,Y+Z],C):- normalize([X,Y]+[X,Z],C).            ...③
normalize([X,Y-Z],C):- normalize([X,Y]-[X,Z],C).            ...③
normalize([X,N*Y],C):- normalize([X,Y],D), C=N*D.           ....④
normalize([X+Y,Z],C):- normalize([X,Z]+[Y,Z],C).            ...⑤
normalize([X-Y,Z],C):- normalize([X,Z]-[Y,Z],C).            ...⑤
normalize([N*X,Y],C):- normalize([X,Y],D), C=N*D.           ....⑥
normalize(X,0):- zero_bracket(X).                           ....⑦
normalize(X,X):- hall_basis(X).                             ....⑧
normalize([X,Y],C):- smaller(Y,X), normalize([Y,X],D), C = -D.  ....⑨
normalize([Y,[X,Z]],C):- (Y=X ; smaller(X,Y)),
   normalize(Y,A), normalize([Y,Z],B), normalize([A,B],C).  ....⑩
```

```
normalize([X,[Y,Z]],C):-
    smaller(X,Y), normalize([Y,[X,Z]]-[Z,[X,Y]],C).        ....⑪
```

In practice, the predicate "zero_bracket" in ⑦ is defined as;

```
zero_bracket([X,X]).
zero_bracket([X,Y]):- zero_bracket(X) ; zero_bracket(Y).
```

We must also add some more rules to simplify the formulas like "2 *[x,y]+3*[x,y]" into "5*[x,y]", but I will not give here further details.

§ 4. Partial computations of the Gelfand-Fuks cohomology of the sphere

The cohomology algebra $H^*(S^n;\mathbb{R})$ is isomorphic to an exterior algebra $E(e)$ generated by one element e of degree n. Suppose V is a two dimensional subspace of V_n with basis $B = \{ x,y ; x < y \}$. Then the Lie subalgebra $E(e)\otimes L(V) \subset E(e)\otimes L(V_n)$ is a retract, and is so on the cohomology level: $H^*(E(e)\otimes L(V)) \subset H^*(E(e)\otimes L(V_n)) \cong H^*_c(\mathcal{L}_{S^n})$.

Now it holds that $E(e)\otimes L(V) \cong L(V)\oplus(e\otimes L(V))$, and denoting $e\otimes L(V)$ by $eL(V)$ we have $C^*(L(V)\oplus eL(V)) \cong C^*(L(V))\otimes C^*(eL(V))$. This last object is bigraded and the differential preserves the bigrading. Thus $H^*(E(e)\otimes L(V)) \cong H^{**}(C^*(L(V))\otimes C^*(eL(V)))$ is bigraded. Similarly for $H^*(E(e)\otimes L(V_n))$, and the retraction explained above is a retraction of bigraded algebras.

By Hilton's theorem (c.f. Haefliger [2]), it holds that $H^{1,0}(C^*(L(V)\otimes C^*(eL(V))) \cong V^\#$ and $H^{0,1}(C^*(L(V))\otimes C^*(eL(V))) \cong (e\otimes V)^\# = eV^\#$, where $^\#$ denotes the dual space. Consequently an irredundant multiplicative generating set of $H^{**}(C^*(L(V))\otimes C^*(eL(V)))$ must contain an additive basis of $H^{1,1}(C^*(L(V))\otimes C^*(eL(V)))/(V^\# eV^\#)$. Due to Haefliger's argument

(see Shibata [3]), the computation of this part of the cohomology is further reduced to that of the cokernel of the differential component

$$d^{(0,1)} : L(V)^{\sharp} \longrightarrow V^{\sharp} \otimes eL(V)^{\sharp} .$$

Since the differential is defined by the Lie product, we can obtain its matrix representation, using the normalization program explained in the preceding section. By computing the rank of the matrix, we obtain the dimension of the cokernel in question in each prescribed degree. This helped me to discover two new infinite families of generators other than those I had previously discovered (Shibata [3]) by hand computations. The homology classes dual to these generators are represented by the cycles as follows.

To simplify the notations, we denote $[x,A]$ by $ad^1(x)A = ad(x)A$, and define $ad^n(x)A$ as $[x, ad^{n-1}(x)A]$ for $n \geqq 2$ (Haefliger's notation). Then for every $n \geqq 2$, our cycles are expressed as

$$z(2,2n) = \sum_{0 \leq i \leq n-2} (-1)^i x \otimes e[ad^i(x)y, ad^{2n-3-i}(x)y] - y \otimes e(ad^{2n-2}(x)y)$$

and

$$\begin{aligned} w(2n-2,2n) = {} & x \otimes e(ad^{2n-3}(y)[x,y]) \\ & - \sum_{0 \leq i \leq n-3} a(i;2n) y \otimes [ad^i(y)[x,y], ad^{2n-4-i}(y)[x,y]] \\ & - y \otimes e(ad^{2n-4}(y)[x,[x,y]]), \end{aligned}$$

where the coefficient $a(i;2n)$ in the above formula is given by

$$a(i;2n) = (-1)^i + (2n-5-2i)\binom{2n-4}{i}/(i+1) .$$

§ 5. Conclusion

Prolog is a computer language especially convenient for symbolic manipulation in pure mathematics. Prolog programs often resembles mathematical formulas. The translation from mathematics to a Prolog program is much easier than in the case of other computer languages. This is mainly due to the following properties of Prolog.

(1) Prolog is of declarative nature. Its program consists of facts and rules.
(2) A statement is expressed in the form of either a fact or a rule, using predicates and their arguments. An arbitrary sequence of letters and symbols beginning with a lower-case letter can be used as a name of a predicate or of an argument.
(3) An arbitrary sequence of letters and symbols beginning with an upper-case letter is a variable. There is no type restriction for Prolog variables. Any kind of mixture of numbers, characters and lists can be a value of a variable.
(4) Program execution starts when the user types in a question, and goes on automatically by the internal inference engine based on the pattern-matching process, which can be interpreted as the execution of Robinson's resolution principle in the axiomatic proof theory.
(5) In practice, recursive definitions are heavily used to remarkably simplify program descriptions (, which, in turn, put a heavy burden to the machine).

With these characteristics, a micro-computer Prolog can be a good handy computational tool for pure mathematicians.

During the present Congress, I tried several micro-computer Prolog softwares. Unfortunately, some of them differ from the standard one in grammer, and others have very small size of stacks to occur stack overflows so easily. But don't be discouraged. There are already good Prolog softwares indeed! And there are also many computer specialists who are

energetically developing Prolog softwares and hardwares.

Prolog programs are easy to write and easy to read in comparison with other computer languages, especially for mathematicians. And they will become still easier to handle in near future.

References

[1] I. Gelfand and D. Fuks, The cohomology of the Lie algebra of tangent vector fields on a smooth manifolds (1), Funcional Analysis, 3 (1969), 32-52

[2] A. Haefliger, Sur la cohomologie de l'algèbre de Lie des champs de vecteurs, Ann. Sci. de l'École Normale Supérieure, 9(1976), 503-532.

[3] K. Shibata, Remarque sur la cohomologie de l'algèbre de Lie des champs de vecteurs sur la sphère, Bulletin de la Société Mathématique de France, 108(1980), 117-136.

[4] K. Shibata, On Haefliger's model for the Gelfand-Fuks cohomology, Japanese Journal of Mathematics, 7(1981), 397-415.

[5] K. Shibata, Applications of the programming language Prolog to linear and homological algebras (3), (in Japanese), Journal of Saitama University, College of Liberal Arts, 4(1986), 43-73.

[6] K. Shibata, Introduction to Prolog for mathematicians, preprint.

Katsuyuki Shibata
Faculty of Liberal Arts,
Saitama University,
Urawa, Saitama, JAPAN 338