Computer Science Journal of Moldova, vol.**4**, no.2(11), 1996

Implementation and usage of the Bergman package shell

A.Colesnikov

Abstract

This article is the survey of author's work on dialog shells over interpreting systems. Aspects of the shell for the computational algebra package Bergman are presented: the solved task, homogenization algorithm, input data checking, approaches to implementation. The shell automatizes and strongly simplifies data preparation and monitoring of the Bergman package.

1 Introduction

There is a lot of popular and frequently used batch or teletype-dialog programs which were embellished by attaching to them corresponding shells. Restricting ourselves by MS-DOS systems only, we have DosShell, Norton Commander (and many others) for MS-DOS, Shez (and many others) for archivers, TEXshell and Scientific Word for TEX, etc.

We present here a shell for a computational algebra package called Bergman [5, 7, 8]. Bergman was developed in the Stockholm University, Sweden, by J.Backelin to calculate the Gröbner basis in the commutative and non-commutative cases. The package was implemented in the Portable Standard Lisp (PSL), firstly under UNIX. Then efforts were applied to further develop the implementation in the cooperative project with the Academy of Sciences of Moldova. The package was successfully transferred to MS-DOS. Now the project is continuing.

Engineering aspects of the Bergman shell are discussed in [1]. Another example of an interpreter shell may be found in [4].

2 Algebraic task solved by the Bergman package

We give here only a brief survey of the algebraic backgrounds. See [2, pp. 27–67, 161–163] for more details and further refernces.

2.1 The non-commutative case

Let $\mathfrak{A} = K\langle X\rangle$ be a free associative algebra with the unity, and let S be the set of all words in the alphabet X. The empty word $\Lambda \in S$ is identified with the unity $\mathbf{1}$.

If $f, g \in S$ then we denote by $\deg_f g$ the number of different occurences of the word f inside the word g. E.g., $\deg_{xx} xxx = 2$, $\deg_{xy} yx = 0$. If $F \subseteq S$ is a set of words then we denote $\deg_F g = \sum_{f\in F} \deg_f g$. In particular, $\deg_X g = |g|$ is the length of the word g.

Let us assume that the set of words S is well ordered, namely:

- every two different words are comparable;
- the induction is possible in order $>$;
- the smallest word is always the unity Λ;
- the order $>$ is preserved after multiplication (1).

$$f \geq g;\ h \geq k \ \Rightarrow\ fh \geq gk;\ hf \geq kg \tag{1}$$

One such order is the *homogeneous lexicographic* order when words are at first ordered by their lengths and words of the same length are ordered lexicographically.

Let us denote by $\hat{u}$ the leading word in the order $>$ of the non-zero element $u \in \mathfrak{A}$. We can now extend the word order $>$ to a partial order on $\mathfrak{A}$: $u, v \in \mathfrak{A}\ u > v \ \Leftrightarrow\ \hat{u} > \hat{v}$. If $U \subseteq \mathfrak{A}$, then we define $\hat{U} = \{\hat{u} : u \in U\}$. Our last definition of degree is $\deg_U v = \deg_{\hat{U}} \hat{v}$.

E.g., $\deg_X v$ is a usual degree (power) of the polynomial v. Another example:

$$\deg_{x^2-x,xy-yx} x^3y - x = 3$$

Let I be an ideal of the free algebra $\mathfrak{A}$.

A word $s \in S$ is called normal relative to the ideal I, if s is not the leading term of any element in I. Let N be the linear hull of the set of normal words. The following theorem is true [2, p. 28]:

Theorem 1 *The following direct sum decomposition of vector spaces holds:* $\mathfrak{A} = N \oplus I$.

Because of this theorem, N is called the *normal complement* of the ideal I. For an element $u \in \mathfrak{A}$, its *normal form* $\bar{u}$ is the image of u by the natural projection $\mathfrak{A} \to N$.

If we define a new operation $\star$ on N by $s \star t = \bar{st}$ then N with this introduced operation is isomorphic to the factor algebra $A = \mathfrak{A}/I$. To work with A within the free algebra $\mathfrak{A}$ it is necessary to be able to find normal words and to reduce any word to its normal form. This problem is algorithmically unsolvable. A very effective approach to its solution in many important cases is the use of the Gröbner basis.

A subset G of the ideal I is called a *Gröbner basis* if $(\forall v \in I)\ \deg_G v > 0$. E.g., the ideal I itself is a Gröbner basis. The following is true:

Theorem 2 *A word s is normal if and only if* $\deg_G s = 0$.

There are many Gröbner bases, however there is always a minimal Gröbner basis no proper subset of which is a Gröbner basis. If G is a minimal Gröbner basis, then $(\forall v \in G)\ \deg_{G \setminus v} v = 0$ (for elements of G, none of leading words is a subword of another leading word). The minimal Gröbner basis is called *reduced* if every $v \in G$ is represented in the form $\hat{v} - \bar{\hat{v}}$. The reduced Gröbner basis is uniquely determined.

Let us assume that the ideal I is generated by the set of elements R. The following three-stage process permits to get a minimal Gröbner basis starting with R.

Step 1: Normalization. Every element $v \in R$ is replaced by its proportional of the form $\bar{v} - w$ (i.e., the coefficient of the leading word is made into 1). After normalizing all the elements from R, perform Step 2.

Step 2: Reduction. Let u and v be such normalized elements that $\deg_v u > 0$. Then $\hat{u} = g\hat{v}h$. The reduction is the substitution of u by the result of the normalization of $u - gvh$. If the reduced element is 0 then we remove it. Otherwise, the reduced element is smaller than u (2).

$$u = \hat{u} - w = g\hat{v}h - w;\ v = \hat{v} - l;\ \Rightarrow u \text{ is replaced by } w - glh \qquad (2)$$

It guarantees the stop of the reduction process. If all possible reductions are fulfilled, perform Step 3.

Step 3: Composition. Let u and v be such normalized elements that some ending of $\hat{u}$ is the beginning of $\hat{v}$. Then $(\exists y \neq 1)\, \hat{u} = xy$, $\hat{v} = yz$. Then we add to R a new element obtained by the normalization of $xv - uz$ (3).

$$u = \hat{u} - w = xy - w;\ v = \hat{v} - l = yz - l;\ \Rightarrow xv - uz = wz - xl \text{ is added} \qquad (3)$$

Even one pair may produce several compositions. If all possible compositions are fulfilled, return to Step 2.

The process may be infinite, but the result is a minimal Gröbner basis. The minimal Gröbner basis can be easily transformed to the reduced one by normalizing its elements and reducing all non-leading words with the aid of the basis itself into their normal form.

2.2 The commutative case

In commutative case, the definition of the Gröbner basis is slightly different because the above defined basis may remain infinite. $\mathfrak{A}$ is now not a free algebra, but the algebra of polynomials $K[x_1, \ldots, x_n]$. Its basis is the ordered set of monomials $x_1^{a_1} \cdots x_n^{a_n}$.

The requirements for the order $>$ are not so strong, namely:

- the unity is the smallest element;
- the order is preserved after multiplication: $f > g \Rightarrow fh > gh$.

The notion of degree is not necessary, and we use divisibility instead. If $A = K[X]/I$ then a normal monomial is a monomial not equal to any of leading monomials of the elements of I. $K[X] = N \oplus I$, where

N is the linear hull of the set of normal monomials. A subset $G \subseteq I$ is called the Gröbner basis of the ideal I if the leading monomial of every element from I is divisible at least by one monomial from the set F of leading monomials of elements from G. A minimal Gröbner basis is always finite in the commutative case.

2.3 The importance of the calculation of the Gröbner basis

The calculation of the Gröbner basis, even the infinite one, permits to solve many algebraic problems and has many applications, e.g., in theoretical physics. Investigations in the area are strongly supported from many sources. To understand it, let us suppose that the we are to solve a system of non-linear polynomial equations $r_i = 0$, where $r_i \in K\langle X\rangle$. Let $R = \{r_i\}$. You can see that the above described process is analogous to the Gaussian method of solving linear equation systems [2, p. 42]. Calculating the the Gröbner basis we can find the exact number of solutions (it is infinite if the algebra $A = \mathfrak{A}/I$ is infinite), and to find solutions themselves. The process may be performed by a computer program. Moreover, in the commutative case the process is always finite.

2.4 Homogeneity, Hilbert and Poincaré series, Anick's resolution

If the factor-algebra A is definer by *homogenoeus* relations, then the elements of a reduced Gröbner basis will be also homogeneous. In that case the algebra A may be presented as the direct sum of homogeneous components $A = \bigoplus_0^\infty A_n$. Homogeneous components A_i correspond to the normal words of the same length, and

$$A_i A_j \subseteq A_{i+j} \tag{4}$$

Such an algebra A is called a *graded* algebra. If all the components A_i are finite dimensional, we can consider the formal series $H_A(t) = \sum_0^\infty (\dim A_i) t^i$, which is called the *Hilbert series* of the algebra A.

For any graded algebra, it is possible to generalize the notion of degree in such a way that the algebra can be obtained from homogeneous relations. If the degree is the word legnth, the graduation is called *natural.* The graded algebra A is supposed to be an assosiative algebra with unity, with 1-dimentional component A_0 generated by the unity.

The calculation ot the Hilbert series is very useful in the infinite (non-commutative) case. Other useful constructions are the Anick's resolution and the Poincaré series. Due to the volume of the corresponding definitions, we can only refer here to [2, pp. 43–62].

In the Bergman package, the defining relations should always be homegenous.

3 The purpose of the shell

The main goal of the Bergman shell was to simplify work with the Bergman package.

The Bergman package has three LISP function of the uppermost level which perform typical calculations in three cases. These three function are composed from about of twenty function of the lower level. In some cases the user may wish to compose his own upper level function from those twenty. There is a lot of internal variables and flags whose values control modes of the calculation.

The used software, Reduce+PSL, is a teletype-dialog system for all platforms. So, to perform a calculation with Bergman, the user is not only to prepare the source data, but to type manually all necessary flags and variables assignments, then, possibly, to enter a new upper level function definition, and, finally, to call the function to solve his problem.

In MS-DOS case, the work was partially automatized with the Bergman shell. Using the shell, the user has advantages of the intuitive and simple interface, when he clicks buttons and check boxes instead of entering LISP commands (see Sec. 4.3 below). The DOS-LISP interaction was implemented in the first shell version through disk files. It was the compromise solution chosen for its simplicity. With it, the possibility to compose own procedures from the intermediate level functions

still exists (see Sec. 4.3, page 274 on the `Task: Run batch` menu item). But the user has no possibility to interact with the system during the LISP calculations.

3.1 Input data checking

The input data for Bergman are the polimomials making up the restriction set. They may be represented in two different forms, named `LISP` and `MAPLE` (algebraic). The output may be presented in three forms, `LISP`, `MAPLE`, and `MACAULAY`. In `MAPLE` and `MACAULAY` forms, the polynomials are distributed, i.e., written without parentheses. The Bergman shell makes some checking of input data.

3.1.1 Commutative case

Suppose we have a commutative polynomial in n variables:

$$P(v_1, \ldots, v_n) = \sum_{i=1}^{r} c_i \prod_{j=1}^{n} v_j^{m_{ij}} \tag{5}$$

where c_1, ..., c_r are integer coefficients, v_1, ..., v_n are variables, and m_{ij} are non-negative integer exponents.

The `LISP` representation of the commmutative polynomial (5) is a list of r elements each of them is a list of $n+1$ integers:

$$((c_1\ m_{11}\ \ldots\ m_{1n})\ \ldots\ (c_r m_{r1}\ \ldots\ m_{rn}))$$

The `MAPLE` representation of the same polynomial in output is

$$c_1\texttt{*}v_1\texttt{\^{}}m_{11}\texttt{*}\ldots\texttt{*}v_n\texttt{\^{}}m_{1n} + \ldots + c_r\texttt{*}v_1\texttt{\^{}}m_{r1}\texttt{*}\ldots\texttt{*}v_n\texttt{\^{}}m_{rn}$$

Occurences of $\texttt{*}v_i\texttt{\^{}0}$ and of `^1` are omitted. `+` is omitted before `-`. If a coefficient $c_i = 1$, it is omitted with the following `*`.

In the `MAPLE` input additional variations are allowed:

- `**` may be used instead of `^`;
- blank space may be inserted, except within identifiers, integers, or the combination `**`;

- exponents 0 and 1 may occur;
- the order of the variables in a monomial is arbitrary.

The variable names $v_1, \ldots, v_n$ may be any valid LISP identifiers. In Bergman implementation, uppercase and lowercase letters are not distinguished. Therefore, to use in a LISP identifier a lowercase letter or a specail sign like `*` or `^` it is necessary to precede such symbol by a `!`.

For example, if we have the following commutative polynomial in 3 variables x, y, and z:

$$x^4 - x^3z - 5xy^2z + 6y^2z^2 - 18xz^3$$

its `LISP` representation is:

```
((1 4 0 0) (-1 3 0 1) (-5 1 2 1) (6 0 2 2) (-18 1 0 3))
```

and one of possible `MAPLE` representations is:

```
x**4-x**3*z-5*x*y**2*z+6*y**2*z**2-18*x*z**3
```

The output `MACAULAY` representation is similar to `MAPLE`, but without `*` and `^`:

```
x4-x3z-5xy2z+6y2z2-18xz3
```

3.2 The polymomial homogeneity

The polynomial is to be homogeneous, that is, the following equalities are to be satisfied (d is the degree of all monomials):

$$d = \sum_{j=1}^{n} m_{1j} = \ldots = \sum_{j=1}^{n} m_{rj} \tag{6}$$

In the commutative case, the non-homogeneous polynomial may be homegenized. Suppose that (6) does not hold:

$$d_i = \sum_{j=1}^{n} m_{ij}\,;\ d = \max(d_1, \ldots, d_r) \tag{7}$$

To homegenize the polynomial, it is necessary to introduce a new variable v_{n+1} and to replace the polynomial $P(v_1, \ldots, v_n)$ by [2, p. 164]:

$$P_1(v_1, \ldots, v_{n+1}) = v_{n+1}^d P(\frac{v_1}{v_{n+1}}, \ldots, \frac{v_n}{v_{n+1}}) \tag{8}$$

The following algorithm (Fig. 1) in Common Lisp [9] may be proposed for a LISP form polynomial to homogenize it.

```
 1   (DEFUN SUMCDR (X) (APPLY '+ (CDR X)))
 2   (DEFUN MINMAX (X) (LIST (APPLY 'MIN X) (APPLY 'MAX X)))
 3   (DEFUN HOMOMONO (MONO)
 4      (SETQ MONO (APPEND MONO (LIST (- DEGREE (SUMCDR MONO)))))
 5   )
 6   (DEFUN HOMOGENIZE (POLYNOMIAL)
 7     (PROGN
 8       (SETQ MM (MINMAX (MAPCAR 'SUMCDR POLYNOMIAL)))
 9       (COND
10         ((= (FIRST MM) (SETQ DEGREE (SECOND MM))) POLYNOMIAL)
11         (T (MAPCAR 'HOMOMONO POLYNOMIAL))
12       )
13     )
14   )

     (setq testpoly1 '((1 2 0)(-2 1 1)(1 0 2)))
                     ; x^2 - 2*x*y + y^2
     (print (homogenize testpoly1))
   ((1 2 0)(-2 1 1)(1 0 2))
   ; The result is equal to the argument

     (setq testpoly2 '((1 2 1)(-2 1 1)(1 0 1)))
                     ; x^2*y - 2*x*y + y
     (print (homogenize testpoly2))
   ((1 2 1 0)(-2 1 1 1)(1 0 1 2))
   ; x^2*y - 2*x*y*z + y*z^2
```

Figure 1:Homogenization of a commutative polynomial in the LISP form

The function SUMCDR (line 1) calculates the sum of the elements of

the list `X` except the first one.

The result of the function `MINMAX` (line 2) is the list of two elements. The first element of the result is the minimal element of the argument list `X`, the second one is the maximal element of `X`.

The function `HOMOMONO` (lines 3–5) homogenizes a monomial `MONO` by appending to the end of its representing list the difference of `DEGREE` and the sum of powers of variables.

The `HOMOGENIZE` function is defined in lines 6–14 of the algorithm. In line 8, the function `SUMCDR` if applied through the standard function `MAPCAR` to all elements of the `POLYNOMIAL` list, obtaining the list of powers of monomials. Then the minimal and maximal powers are obtained by `MINMAX`. The conditional calculation checks firstly the equality of minimal and maximal of monomial powers (line 10). As a side effect, the maximal monomial power is assigned to the variable `DEGREE`. If the equality holds, the argument `POLYNOMIAL` is returned untouched as the result. Otherwise (line 11) the function `HOMOMONO` is applied to each monomial of the argument, and the list of resulting monomials with one additional element is the final result.

For simplicity, some details of real algorithm are omitted in line 11. They are the generation of a name for a new variable, the notification for the user, the incrementing of the internal variable `EMBDIM` and other corresponding changes.

The algorithm was implemented on a computer. Example results are shown on Fig. 1.

3.2.1 Non-commutative case

In the non-commutative case, the following homogeneous polynomial of degree d

$$P = \sum_{i=1}^{r} c_i \prod_{j=1}^{d} v_{ij} \tag{9}$$

where $v_{ij} \in X$ are variables, is represented in the `LISP` form as a list of lists of integers of length $d + 1$:

$$((c_1\ m_{11}\ \ldots\ m_{1d})\ \ldots\ (c_r m_{r1}\ \ldots\ m_{rdn}))$$

Here $1 \leq m_i j \leq n$ are indices of variables in their order by increasing significance in the list $(v_1 \ \ldots \ v_n)$.

The `MAPLE` representation of the same polynomial in output is

$$c_1\texttt{*}v_{11}\texttt{*}\ldots\texttt{*}v_{1d} + \ldots + c_r\texttt{*}v_{r1}\texttt{*}\ldots\texttt{*}v_{rn}$$

Occurences of $\texttt{*}v_i\texttt{\^{}0}$ and of `^1` are omitted in `MAPLE` output. `+` is omitted before `-`. If a coefficient $c_i = 1$, it is omitted with the following `*`.

In `MAPLE` input format, the identical sequential factors may be collected and represented through exponents. E.g., if the order of three variables is `(x y z)`, and the `LISP` form is `((5 1 1 1 3 2 2 1))`, then the `MAPLE` output is `5*x*x*x*z*y*y*x`, but it may be typed in input as `5*x**3*z*y**2*x`.

4 The presentation and behavior of the Bergman shell

4.1 The shell running loop

The Bergman shell is launched from the PSL interpreter. The sequence is as follows:

1. The Reduce interpreter is started consuming a standard script as the input program. This input script contains Reduce commands switching to PSL mode, the LISP function `DOSSHELL` definition and the infinite LISP loop calling this function.
2. The PSL calls `DOSSHELL`.
3. Through the MS-DOS PSL extension – the `SYSTEM` function – the MS-DOS shell executive module `BERGM_SH.EXE` is started under MS-DOS. It is important that in the moment the PSL interpreter, the compiled image of the BERGMAN package and all memory allocated for them is used. We have MS-DOS and the Bergman shell in the lower memory and the package waiting in the upper (extended) memory. For all shell operation we have no more than 400 Kbytes.

4. The shell is utilized by the user to prepare LISP input files with function calls, and data files with polynomials.
5. The shell finishes its work writing prepared files to the disk and rerouting the PSL interpreter input from that file containing function calls.
6. The interpreter interprets newly created file and calls functions solving the algebraic task. The LISP teletype-mode output is seen on screen.
7. The interpreter input from files is nested. Finishing the algebraic calculations, it returns to the infinite loop calling shell (step 2).
8. To break the loop, the user may, within the shell, press the `Alt`+`X` key combination or to click with the mouse the `Exit` menu item. The shell writes to the disk the program of the single LISP command '`(QUIT)`'. This command is interpreted on step 6 and forces the loop break by simply stopping the interpreter. (During the LISP interpretation, the user may press `Ctrl`+`C` to stop.)

4.2 Usual menu items

The Bergman shell was implemented using Borland Pascal with Objects 7.01 and its supplied package Turbo Vision [6]. Its external representation is usual for such programs.

You see on the screen the shell desktop between the menu bar at the top and the status line at the bottom. The menu items may be selected by keys or mouse. We have six usual menu items (`File`, `Edit`, `Search`, `Options`, `Window`, and `Help`), and three specific items (`Task`, `View`, and `Additions`).

Through the `File` menu item we can open files for editing, saves files, change the current directory, exit temporarily to MS-DOS, etc. The `File: Exit` item means writing the LISP program '`(QUIT)`' as the next interpreter input before terminating the shell (see step 8 above). The exit from the shell to LISP calculations (not stopping the interpreter) is performed from another item (see below).

We had used a standard Turbo Vision implementation of the `File`

item. Our experience shows that in many cases the desired extension to this submenu is `File: Erase` to delete a file from the disk.

Other usual menu items do not need any comment.

4.3 Specific menu items and their functions

To define the solved task, the user is to pass three menu items – `Task: Options`, `Task: Flags`, and `Task: In/Out`.

When the user selects the `Task: Options` menu item, he sees the Options dialog. This dialog contains the calculation defining panel – the Commutativity radio buttons, the Strategy radio buttons, and the Poincaré-Betti and Hilbert series check box. Combining these setting, the user selects one of the three possible calculations:

`Simple` Commutative or non-commutative algebra, ordinary Buchberger's algorithm, no Poincaré-Betti and Hilbert series.

`StagSimple` Commutative algebra, staggered linear basis algorithm with substance (SAWS), no Poincaré-Betti and Hilbert series.

`NcPBH` Non-commutative algebra, Buchberger's strategy, double Poincaré-Betti and Hilbert series of the associated monomial ring calculation.

For the `Simple` calculation the user is to select the Ordinary Strategy not checking PBH Series box. He can select any Commutativity – commutative with lexicographical order, commutative with reverse lexicographical order, and non-commutative.

For the `StagSimple` calculation the user is to select the SAWS Strategy, commutative (lex or rev-lex) Commutativity not checking PBH Series box.

For the `NcPBH` calculation the user is to select the Ordinary Strategy, non-commutative Commutativity, checking PBH Series box.

The dialog automatically blocks illegal calculation option combinations. To change previously set options, the user may need to mark them in some order. E.g., if he had solved the `StagSimple` problem before, he had marked commutative (lex or rev-lex) and SAWS buttons, not marking PBH Series box. In the case the non-commutative button and PBH Series box are disabled, and the user can not mark

them. To select the non-commutative variant, he marks at first the Ordinary Strategy – then non-commutative button will be enabled (but PBH Series box remains disabled until the user does select the non-commutative radio button).

The selected calculation mode is indicated in the lower-right corner of the screen (in the status line).

Other elements of the `Task: Options` dialog are independent and may be selected in any order.

The `Task: Flags` dialog permits to define desired flags. Depending of the calculation mode, some flags may be disabled. Except specific, there are non-specific flags, e.g., if the user wants verbose garbage collection, he marks `Verbose GC` checking box. It corresponds to the standard PSL flag '`GC`'.

The last dialog is the `Task: In/Out` dialog where the user points to or enters input and output information. Depending of the calculation mode, there are three different dialogs – `Simple` defines only one output result, `StagSimple` – two, and `NcPBH` – three. Results may be directed to the file, or to the screen. The input may be written directly on the screen, or taken from the file. To define this the user selects the corresponding button and then presses the [→] key, or clicks the onscreen → with the mouse.

In any file selection dialog the user has the possibility to open file for editing. Closing the corresponding edit window he returns to the same file selection dialog.

Having selected the input and output modes and entering data the user can press [Enter] of click the Run button in the `Task: In/Out` dialog to start the Bergman calculation. It means that the shell writes on the disk the LISP program. E.g., the program contains a line '`(ON flag_name)`' or '`(OFF flag_name)`' for each flag `flag_name` meaningful for the chosen calculation mode, depending of the settings in the `Task: Flags` dialog. Then the shell reassigns the PSL input from the formed file and terminates.

When the calculation is finished and the shell is reentered, the user can view the LISP echo screen pressing [Alt]+[F5], and view the output results through the `View` menu item.

The user can prepare the whole task in a single file and run it through the `Task: Run batch` menu item.

The `Additions` menu item calls additional programs not included in the Bergman package. One such program is the Anick's resolution calculation [3]. The algorithm was developed by V.Ufnarovsky and implemented in C++ by postgraduate student A.Podoplelov. It is the test variant which had shown satisfactory results (see [8]) and is supposed to be rewritten in LISP to be included into the Bergman package itself. The rewriting would enlarge available memory, would permit to use intermediate results and the Gröbner basis from the LISP memory, and has other advantages.

The Anick's resolution program has its own parameters and command language. The `Additions: Anick's resolution` dialog has an additional window showing the formed command sequence whilst the user clicks buttons and fills input fields. E.g., if the user presses the `Derive all chains` button and then fills the input order field with the number 5, he sees in the command sequence the '`d 5`' command added to calculate derivations of order 5 chains. The user may edit the formed command sequence directly in its window. This property was found a very useful and demonstrative one so it is planned to implement such window for the main formed LISP program.

Acknowledgments

The author thanks J.Backelin, S.Cojocaru, J.-E.Roos, and V.Ufnarovski for their help and fruitful discussions.

This work was partially supported by the Royal Swedish Academy of Sciences, project number 1502, and by the International Association for the promotion of cooperation with scientists from the independit states of the former Soviet Union, grant number INTAS-93-0030.

References

[1] A.Colesnikov, L.Malahova. The Bergman package shell: an example of interface to the interpreting system. // Computer Science Journal of Moldova, vol. **3**, nr. 2 (8), 1995.

[2] V.A.Ufnarovskij. Combinatorial and Asymptotic Methods in Algebra. / In: A.I.Kostrikin, I.R.Shafarevich (Eds.). Algebra VI. – Encyclopaedia of Mathematical Sciences, v. 57. – Berlin, Heidelberg, New York: Springer-Verlag, 1995.

[3] D.Anick. On the homology of associative algebras. // Trans. Am. Math. Soc., **v. 296**, nr. 2, 1986, pp. 87–112.

[4] A.Colesnikov, L.Malahova. A portable interpreter shell and its implementation for a perspective computer. / In: Computer software and programming (Mathematical Investigations, **issue 115**). Chişinău, 1990, pp. 92–96 (in Russian).

[5] J.Backelin, R.Fröberg. How we proved that there are exactly 924 7-roots. / S.M.Watt, ed. Proc. ISAAC'91, ACM, 1991, pp. 103–111.

[6] Borland Pascal with Objects 7.0. Turbo Vision Version 2.0 Programming Guide. – Borland International, Inc., 1992.

[7] J.-E.Roos. A computer-aided study of the graded Lie algebra of a local commutative Nötherian ring. // J. Pure Appl. Algebra, **v. 91**, 1994, pp. 255–315.

[8] S.Cojocaru, V.Ufnarovski. Noncommutative Gröbner basis, Hilbert series, Anick's resolution and BERGMAN under MS-DOS. // Computer Science Journal of Moldova, **v. 3**, nr. 1 (7), 1995, pp. 24–39.

[9] Guy L. Steele. Common Lisp the Language. – 2nd edition . – Digital Press: Woburn, MA, 1990.

[10] Borland® Pascal with Objects. Version 7.0. User's Guide. - Borland International, Inc., Scotts Valley, CA, 1991.

Alexander Colesnikov (b.1947) is a scientific researcher at the Institute of Mathematics of Academy of Sciences of the Moldova Republic. He had graduated from the Moscow State University. His scientific interests include compiler construction, computational algebra, natural language processing, computer text processing and computer typography, object-oriented programming, software engineering, man-computer interfaces. He had published 35 scientific papers and co-authored of the manual "LaTeX prin exemple" ("LaTeX by example", in Romanian).
Phone: (373-2) 738058
E-mail: kae@math.moldova.su