数理解析研究所講究録
第 581 巻 1986 年 105-130

An Implementation of the Formula Manipulation Package in Common Lisp

by

Morio Nagata and Masako Abe

永田 守男　　阿部 昌子

Department of Administration Engineering

Faculty of Science and Technology

Keio University

（慶応義塾大学）

3-14-1 Hiyoshi, Yokohama

JAPAN

ABSTRACT

We propose a simple formula manipulation package called MMP including a certain kind of modern algebra through the package system of Common Lisp.

Two main features of this work are the following. First, we have provided an interface including various facilities. This interface enables that the user easily utilize some parts of formula manipulation functions when developing new Lisp programs. Second, we have developed a method for constructing the formula manipulation system of modern algebra by using Groebner bases and algebraic extension theory.

We have implemented the package manipulating multivariate polynomials and rational expressions on a certain types of rings. The effectiveness of our package have been certified with the experimental use of our colleagues.

1. Introduction

Well-qualified formula manipulation systems have already been implemented [1,2,3]. However, they are so called "closed systems" to Lisp programmers, that is, it is difficult to utilize some parts of those facilities when developing new Lisp programs. Though several formula manipulation packages have been implemented for programs of numeric computation [4,5], there are no packages for programs of symbolic computation.

In this paper, two terms "formula manipulation system" and "formula manipulation package" are used in the following sense. A formula manipulation system is the software which is directly used by the human user, and a package can be used by the program written in the language of the user's choice [5]. The formula manipulation system tends to be a large and general system for many users. However, there exist Lisp programmers using a simple and special package.

Many Lisp programs, i.e. an automatic theorem prover, an automatic program verifier, an expert system etc., need formula manipulation functions [6]. Though a lot of formula manipulation systems are written in the Lisp language, the Lisp programmer has sometimes to write programs manipulating formulas when developing new systems.

We propose a formula manipulation package called MMP (Modular Mathematical Package) to be widely used in the Lisp language. In this paper, first we introduce an overview of our package by using simple examples. Next, we show the design principles and implementation methods of the package.

This package has two main features. First, MMP provides a flexible interface with both a human user and another Lisp program. Second, the user can specify some algebraic structures, especially cetain rings and fields in MMP. By using Groebner bases, MMP manipulates polynomials on the specified structure. For this implementation, we have developed an effective method for constructing Groebner bases.

MMP is implemented through the package system of Common Lisp [7]. If it is slightly modified, it will be an algebraic package for other programming languages.

2. An Overview of Our Package

We introduce an overview of MMP by using simple examples. There are two aspects in this package. One is a simple formula manipulation package for symbolic computation. The other is that MMP manipulates formulas on algebraic structures, especially rings and fields. We explain the former at first, the latter next.

From the point of view of the former aspect, MMP seems to be a traditional formula manipulation package computing polynomials and rational expressions.

When we use facilities of MMP in the Lisp program, we write

```
        (function-name arguments).
```

The last character of every function name of MMP is an asterisk (*).

For example, the addition of polynomials is written as

```
     (add* '(x+1) 'y '(z^2+3)).
```

The value of this expression is

```
        (x + y + z ^ 2 + 4).
```

Since we assume that various types of programs use MMP, four forms of mathematical expressions can be accepted (cf. 5.1). Moreover, it accepts arguments of a function in different forms. However, we show all examples of this chapter in infix forms.

MMP provides functions of subtraction, multiplication, division, substitution, differentiation, factorization etc. for polynomials and rational expressions. Note that these functions can be used by both human users and other Lisp programs. A human user can use these functions through the user interface of this package (cf. Appendix II).

Now, let us show the latter aspect of our package. In this case, the main algebraic structure of this system is a ring, especially a polynomial ring. Thus the fundamental function is to define a particular ideal of a ring.

For example, if we write

```
(set-ideal* 'masako '(5)),
```

then all operations are perfomed under modulo 5 and the ideal can be used with the name "masako" thereafter. Therefore, the result of

```
(mult* 'a '(a+c) '(b2*a^3-4*b*a^5))
```

is

```
(b * a ^ 5 + b ^ 2 * a ^ 3 + 2 * a + c)
```

in this situation.

When we reset this ideal, we write

```
(clear-ideal* 'masako).
```

Next, if

```
(set-ideal* 'masako '((x) (y+1)))
```

is given, then our package computes Groebner bases of x and y+1 [8]. In this case, these two polynomials are Groebner bases. Therefore, x and y+1 are in this ideal. In this situation, the result of

```
(mult* 'd '(y-1) '((a+2)^3))
```

is

```
(- 2 * d * a ^ 3 - 12 * d * a ^ 2 - 24 * d * a - 16 * d).
```

The function computing Groebner bases of polynomials can also be used in MMP. The details will be shown in 4.2. We describe our method for constructing Groebner bases in 4.3.

The real record of some examples will be shown in Appendix I. Furthermore, a user interface is provided in MMP. When a person uses MMP, he or she can use functions of the package through this interface. A session example will be shown in Appendix II.

3. Functions on Polynomials and Rational Expressions

MMP provides computing facilities on rings of modern algebra [9]. Furthermore, it can be used as a traditional formula manipulation package to perform usual operations on multivariate polynomials and rational expressions. We describe traditional facilities here.

Let both a and b be multivariate polynomials. Then, addition, subtraction and multiplication of a and b are written as (add* a b), (subtract* a b) and (mult* a b) respectively. (div* a b) returns the list of the quotient and the remainder of a/b. Greatest common divisor and least common multiplier of a and b are written as (gcd* a b) and (lcm* a b) respectively. If a and b are equal to each other in mathematical expressions, then (equal* a b) returns T. Otherwise it returns NIL. Notice that the number of arguments of any one of these functions is an arbitrary non-negative integer.

The multiplication of n-times of a is written as (power* a n). (diff* a x) represents the partial differentiation of a on the variable x. If we substitute a for the variable x in the expression b, then we write (subst* a x b).

Now, let both a and b be rational expressions. Addition, subtraction, multiplication and equality test are written in the same forms of polynomials. (div* a b) returns the rational expression of a/b.

4. Facilities on Rings and Fields

4.1 Groebner Bases and Ideals

It is expected that a modern formula manipulation system should have facilities on modern algebra. Above all, facilities on rings and fields are useful for manipulating multivariate polynomials. Moreover, there are many applications manipulating multivariate polynomials on a certain residue ring [10].

Groebner bases are useful for manipulating polynomials on a ring in the computer algebra system. Especially, if we use the Groebner basis, we can manipulate polynomials on a certain residue ring by a particular ideal in a canonical form. Therefore, MMP provides a function groebner* for computing the Groebner basis. For example,

```
        (groebner* '(x*y^2+1) '(x*y+x^2))
```

returns

```
        ((x ^ 3 + 1) (x + y)).
```

When MMP accepts a set of multivariate polynomials for an ideal, it produces the Groebner basis by using groebner*. It manipulates formulas on the residue ring by this ideal thereafter.

4.2 Calculations on Rings and Fields

Roughly speaking, MMP manipulates polynomials on a certain residue ring of polynomials. The user can specify one of three types of residue rings by indicating its ideal. The following types are provided in this package. The first is calculation on Z_p; The second is on Q and its algebraic extension field; The third is on Z_p and its algebraic extension field.

Let us consider polynomial rings here. In this case, specifying an ideal is to determine the quotient ring. If nothing is specified as an element of the ideal, then MMP assumes that all formulas are manipulated in Q. This is the normal case.

When a prime number p is given, then all computations of polynomials are performed on Z_p. This case is called the first computation type in this paper.

Elements of the algebraic extension of Q can be determined by specifying multivariate polynomials as elements of the ideal. This is called the second computation type.

If a prime number p and multivariate polynomials are given, then Z_p and its algebraic extension field is fixed. This is called the third computation type.

In our package,

```
(set-ideal* 'nag '(7))
```

specifies the first computation type. All formulas are manipulated by modulo 7. If we write

```
(more-ideal* 'nag '((x)(y+1))),
```

then all polynomials are manipulated by the ideal 7, x and y+1. In the above three computation types, MMP manipulates formulas based on the Groebner basis. Thus, every formula is represented in a canonical form.

The following three functions modify the elements of ideals:

```
set-ideal*, clear-ideal* and more-ideal*.
```

When we specify a particular ideal, we write

```
(ideal* name-of-ideal).
```

The following will show the information on the state of the ideal to the user.

current-ideal$, ideal-element$

base$ and ideal-list$

4.3 A Method for Computing Groebner Bases

Constructing the Groebner basis is a time-consuming task. When the number of elements of an initial set of multivariate polynomials is large or the degrees of those polynomials are high, the computation may be impractical even on big computers. Thus, we have developed a method for computing the basis in an effective way by modifying Buchberger's algorithm. This section describes an outline of our method with simple examples. We use notations and terms given in Buchberger's survey [8].

The basic algorithm for constructing Groebner bases requires the executions of two subalgorithms called "Spolynomial" and "Normalform" for all pairs of a set of multivariate polynomials. The elements of the initial set are given by the user. The number of the elements of the set sometimes increases during the execution of this algorithm. Moreover, "Normalform" is required to compute the reduced Groebner basis.

Now, let us consider to modify the basic algorithm for constructing Groebner bases effectively.

First of all, we pay attention to the order in the set of multivariate polynomials $F=\{f_1,f_2,\ldots,f_n\}$. We assume all elements of F satisfy

$$\text{Normalform}(f_i,(f_{i+1},\dots,f_n))=f_i \quad (1\leq i<n).$$

Using this order, we write an n-tuple $(f_1,f_2,\dots,f_n)$ instead of the set F. Moreover, we assume that Normalform generates the similar results for both the set and the tuple. Groebner basis of the tuple is also considered as the similar form of the set.

We define an algorithm "Semireduce" reducing an n-tuple $F=(f_1,\dots,f_n)$ to another tuple F' such that GB(F)=GB(F') and $\#(F)\geq\#(F')$.

<u>Algorithm</u> 1 (F':=Semireduce(F)):

```
F' := $(f_1)$
F := $(f_2,\dots,f_n)$
i := 2
while $i\leq n$ do
    h := Normalform($f_i$,F')
    F := $(f_{i+1},\dots,f_n)$
    i:= i+1
    if $h \neq \emptyset$ then
        F' := cons(h,F')
```

where cons(h,F') is the tuple adding the element h to the head of the tuple F'.

Now, by using Semireduce, we can modify the basic algorithm as follows. We assume that a set of multivariate polynomials in the form of the tuple. The order in the given tuple has no meanings, that is, the initial tuple is considered as the set.

```
Algorithm 2 (G:=GB(F)):
        G := F
        B := {(f_i,f_j)|f_i,f_j∈F,i≠j}
        while B≠∅ do
            (f_i,f_j) := a pair in B
            B := B-{(f_i,f_j)}
            h := Spolynomial(f_i,f_j)
            h' := Normalform(h,G)
            if h'≠∅ then
                G := cons(h',G)
                G := Semireduce(G)
                B := {(g_i,g_j)|g_i,g_j∈G, i≠j}
```

In constructing G of this new algorithm, one step reduction must be

$$g_i \xrightarrow[g_j]{1} g'_i \qquad (j<i).$$

Therefore, we need not consider the steps for $j\geq i$.

For example, if we use the basic algorithm for constructing the reduced Groebner basis, then an initial set of three multivariate polynomials

$$3x^2y+2xy+y+9x^2+5x-3$$
$$2x^3y-xy-y+6x^3-2x^2-3x+3$$
$$x^3y+x^2y+3x^3+2x^2$$

requires the executions of "Spolynomial" and "Normalform" 28 times and 35 times respectively.

On the other hand, if we give the same set for constructing the reduced Groebner basis, then the set requires those executions are 5 times and 14 times by using this new algorithm.

5. Implementation

5.1 Organization of Our Package

MMP is written in the VAX LISP language which is an extended implementation of the Common Lisp language [11]. MMP consists of four parts. These parts and the flow of control are shown in Figure 1.

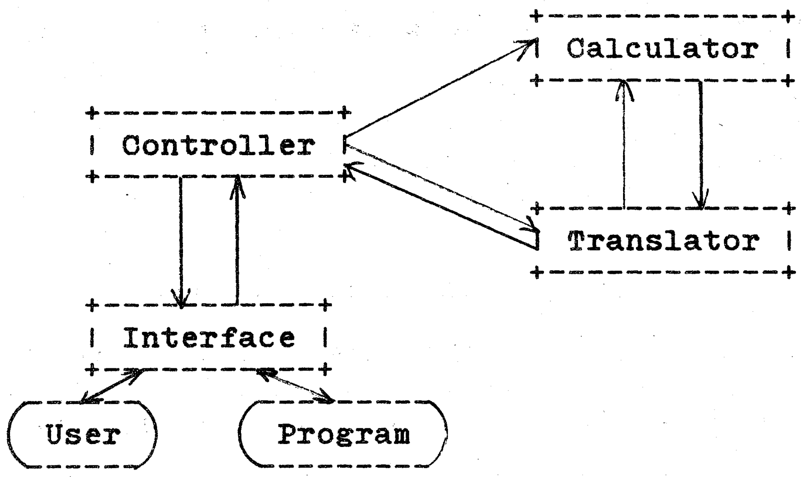

Fig. 1 Components of Our Package

Controller controls other three parts. Interface accepts the requirements of the human user or the Lisp program, and gives the results of computation. Calculator performs operations on formulas. Translator translates formulas represented in arbitrary forms prepared in our package into canonical forms. MMP accepts formulas represented in one of four forms, i.e. infix, prefix, recursive and expanded forms. We explain these forms by using the following expression.

$$\frac{x+1}{x^2y^2+3xy^2+5}$$

The infix form of this expression is

```
(x+1)/(x^2*y^2+3*x*y^2+5).
```

Its prefix form is

```
(/ (+ x 1) (+ (* (^ x 2) (^ y 2))
              (* 3 x (^ y 2))
              5)).
```

Above forms are mainly used by the human user or the Lisp program. The recursive form

```
((x 1 1 0 1) (y 2 (x 2 1 1 3) 0 5))
```

is the internal representation of MMP. Calculation excepting the computation of Groebner bases is performed on expressions in this form. The expanded form

```
(((x) 1 (1) 1 (0))
 ((y x) 1 (2 2) 3 (2 1) 5 (0 0)))
```

is used in the computation of Groebner bases. These two forms are internal representations of MMP. But the human user and the program can use expressions in these forms.

5.2 Structure of Calculator

According to the expressions and operations, our calculator automatically performs a suitable calculation. Table 1 shows the relation of operational objects and calculation. For instance, when ordinal multivariate polynomials and usual computation is required, this calculator assumes that the element of the ideal is null and the computation is performed without Groebner bases.

Table 1 Operational Objects and Calculation

Object	Elements of Ideals	Groebner Bases
Multivariate Polynomials and Rational Expressions	∅	x
Ring of Multivariate Polynomials with Z_p Coefficients	Prime Number	x
Residue Ring of Multivariate Polynomials with Q Coefficients	Polynomials	Polynomials with Q Coefficients
Residue Ring of Multivariate Polynomials with Z_p Coefficients	Prime Number + Polynomials	Polynomials with Z_p Coefficients

5.3 Interface with Programs and Users

We intend that MMP is used by both a human user and another Lisp program. Therefore, it has an interface including various facilities. These facilities are devided into three types.

The first type is a human interface for novice users. This is an interactive and menu-driven interface [12]. If a person uses MMP through this interface, he or she can perform own work only by following the instructions of this system. A session example of this type will be shown in Appendix II.

The second is an interface translating any form of a formula into another form. MMP has four forms described in 5.1. Now, let $f_1, f_2, \ldots, f_n$ be mathematical expressions represented in forms shown in 5.1 ($n \geq 1$). Then

```
(to-i* f1 f2 ... fn)
```

returns the list of expressions represented in infix forms. Both the recursive form and the expanded form have the ratinal expression type and polynomial type. For example,

```
(to-r-rational* '(x+y*2-7*s^3/r))
```

returns

```
((y 1 (r 1 2) 0 (x 1 (r 1 1) 0 (s 3 -7)))(r 1 1)).
```

On the other hand,

```
(to-r-poly* '(x+y*2-7*s^3))
```

returns

```
(y 1 2 0 (x 1 1 0 (s 3 -7))).
```

Moreover, MMP has two functions, to-e-rational* and to-e-poly*, in a similar manner.

Notice that every function of MMP can accept formulas in all forms of these types. Moreover, it accepts formulas in different forms as arguments of a function. Thus, we can write the following expression.

```
(add* '(+ a b) '(c+d))
```

The result of this expression will be shown as

```
(+ d c b a).
```

If we write

```
(add* '(a+b) '(+ c d)),
```

then the result will be shown as

```
(d + c + b + a).
```

The third type of facilities of the interface is that the user can write own Lisp programs without worrying about name conflicts in their programs and MMP. This is owing to the package system of Common Lisp.

6. Concluding Remarks

Two colleagues wrote Lisp programs using our program before packaging it. In this experimental use, there existed some name conflicts of functions and variables. Thus, we have convinced that the package system is useful for our purpose.

When we think of the practical use of MMP, the enhancement of facilities is needed. Finally, it is expected that MMP-like packages are widely implemented and used.

References

[1] Hearn, A.C.: Reduce User's Manual, Version 3.1, The Rand Corporation, Santa Monica, Cal., 1984

[2] Jenks, R.D.: A Primer: 11 Keys to New Scratchpad, Proceedings of EUROSAM 84, Lecture Notes in Computer Science No. 174, Springer-Verag, Berlin, 1984, pp. 123-147

[3] Bogen, R.A., et al.: MACSYMA Reference Manual, Version 6, MIT, Cambridge, Mass., 1977

[4] Brown, W.S., Tague, B.A., Hyde, J.P.: The ALPAK System for Numerical Algebra on a Digital Computer, Bell Syst. Tech. J., Vol. 42, 1963, 2081-2119

[5] Shearer, J.M., and Wolfe M.A.: Algib, a Simple Symbolic-Manipulation Package, CACM, Vol. 28, No. 8, 1985, pp. 820-825

[6] Calmet, J., and van Hulzen, J.A.: Computer Algebra Applications, Computing, Suppl. 4, 1982, pp. 245-258

[7] Steele, G.L. Jr.: Common LISP: The Language, Digital Press, Bedford, Mass., 1984

[8] Buchberger, B.: A Survey on the Method of Groebner Bases for Solving Problems in Connection with Systems of

Multivariate Polynomials, Proc. RSYMSAC, Riken, Wako-shi, Japan, 1984, pp. 7-1 - 7-15

[9] van der Werden: Modern Algebra I, Springer, Berlin, 1971

[10] Gilbert, W.J.: Modern Algebra with Applications, John Wiley and Sons, New York, 1976

[11] Digital Equipment Corp.: VAX LISP User's Guide, DEC, Maynard, 1984

[12] Nagata, M., and Shibayama, M.: An Interactive Algebraic System for Personal Computing, Proc. of IEEE International Symposium on New Directions in Computing, Trondheim, Norway, 1985, pp. 130-137

Appendix I

A Real Record of Using MMP

```
[ in-put ]current-ideal$
SYSTEM-IDEAL$

[ in-put ](get current-ideal$ 'calculation-type)
RATIONAL

[ in-put ](add* '(x * 4 - 3) 'v '((w e) 2 (1 0) 1 (0 7)))
(4 * X + 2 * W + V + E ^ 7 - 3)

[ in-put ](subtract* 'x 4 'y)
(- Y + X - 4)

[ in-put ](div* '(x ^ 2 + 1) 'x)
(X 1)

[ in-put ](div* '(x ^ 2 + 3 / d) 'x)
((D * X ^ 2 + 3) / (D * X))

[ in-put ](power* '(x / y) 4)
((X ^ 4) / (Y ^ 4))

[ in-put ](diff* '(x ^ 4 - 7 / x ^ 2) 'x)
((4 * X ^ 6 + 14) / (X ^ 3))

[ in-put ](subst* '(y + s) 'x '(x * 3 - y * 2))
(Y + 3 * S)

[ in-put ](gcd* '(x + 1) '(x ^ 2 + 2 * x + 1) 0)
(X + 1)

[ in-put ](lcm* '(x + 1) '(x ^ 2 + 2 * x + 1) 0)
0

[ in-put ](equal* '(^ a 3) '(a ^ 3) '((a) 1 (3)) '(a 3 1))
T
```

```
[ in-put ](set-ideal* 'masako '(5))
MASAKO

[ in-put ](get 'masako 'calculation-type)
MODULO-P

[ in-put ]base$
5

[ in-put ](add* 2 3 4 5 6 7 8)
0

[ in-put ](add* 'a '(+ a c) '((b a) 1 (2 3) -4 (1 5)))
(+ C (* (^ A 3) (^ B 2)) (* (^ A 5) B) (* 2 A))

[ in-put ](subtract* 'a 'b 'c)
(4 * C + 4 * B + A)

[ in-put ](mult* '(x + y)  7 '(/ r 3))
(4 * R * Y + 4 * R * X)

[ in-put ](div* '(a * b * c) 'b 'c)
(A 0)

[ in-put ](div* '(a * b - c) 'c)
(4 (A * B))

[ in-put ](diff* '(x ^ 8 - 3 * x) 'x)
(3 * X ^ 7 + 2)

[ in-put ](subst* '(y - 1) 'x '(x ^ 2 + 3))
(Y ^ 2 + 3 * Y + 4)

[ in-put ](gcd* '(x + 7) '(x ^ 2 + 14 * x + 49))
(X + 2)

[ in-put ](lcm* '(x + 7) '(x ^ 2 + 14 * x + 49))
(X ^ 2 + 4 * X + 4)

[ in-put ](equal* 2 3 4)
NIL

[ in-put ](equal* 0 '(* x 5))
T
```

```
[ in-put ]ff
(3 * X ^ 2 * Y + 2 * X * Y + Y + 9 * X ^ 2 + 5 * X - 3)

[ in-put ]gg
(2 * X ^ 3 * Y - X * Y - Y + 6 * X ^ 3 - 2 * X ^ 2 - 3 * X + 3)

[ in-put ]hh
(X ^ 3 * Y + X ^ 2 * Y + 3 * X ^ 3 + 2 * X ^ 2)

[ in-put ](set-ideal* 'nag (list ff gg hh))
NAG

[ in-put ]current-ideal$
NAG

[ in-put ](get current-ideal$ 'calculation-type)
MODULO-MULT

[ in-put ]base$
((Y 1 1 0 (X 2 1 1 -3/2 0 -3)) (X 3 1 2 -5/2 1 -5/2))

[ in-put ]ideal-element$
((3 * X ^ 2 * Y + 2 * X * Y + Y + 9 * X ^ 2 + 5 * X - 3) (2 * X ^ 3 * Y - X * Y
- Y + 6 * X ^ 3 - 2 * X ^ 2 - 3 * X + 3) (X ^ 3 * Y + X ^ 2 * Y + 3 * X ^ 3 + 2
 * X ^ 2))

[ in-put ](add* ff gg hh)
0

[ in-put ](subtract* '(x ^ 5) '(y ^ 2))
(245/8 * X ^ 2 + 113/8 * X - 9)

[ in-put ](subst* '(x ^ 5 + 1) 'a '((a + 4) ^ 7))
(69791519438415336730351 5625/8589934592 * X ^ 2 + 534299052914816561074609375/8589934592
 * X + 78125)
```

```
[ in-put ](set-ideal* 'mn (list 19 ff gg hh))
MN

[ in-put ]current-ideal$
MN

[ in-put ](get current-ideal$ 'calculation-type)
MODULO-MULT&P

[ in-put ]base$
(19 (Y 1 1 0 (X 2 1 1 8 0 16)) (X 3 1 2 7 1 7))

[ in-put ]ideal-element$
(19 (3 * X ^ 2 * Y + 2 * X * Y + Y + 9 * X ^ 2 + 5 * X - 3) (2 * X ^ 3 * Y - X *
 Y - Y + 6 * X ^ 3 - 2 * X ^ 2 - 3 * X + 3) (X ^ 3 * Y + X ^ 2 * Y + 3 * X ^ 3 +
 2 * X ^ 2))

[ in-put ](add* 13 '(x ^ 9 - 7) '(a ^ 5 * 183))
(12 * X ^ 2 + 15 * X + 12 * A ^ 5 + 6)

[ in-put ](subtract* '(d + x ^ 3 - 7) 3 4 'y)
(13 * X ^ 2 + X + D + 2)

[ in-put ](gcd* '(x * y ^ 2 + 1) '((x * y ^ 2 + 1) ^ 2))
1

[ in-put ](add* '(x * y ^ 2 + 1))
(11 * X ^ 2 + 17 * X + 1)

[ in-put ](power* (add* '(x * y ^ 2 + 1)) 2)
(X ^ 2 + X + 1)
```

Appendix II

A Session Example of MMP

```
[ in-put ](top*)


        Welcome to our package!

 We can use following functions.

        add*           gcd*           groebner*
        subtract*      lcm*           set-ideal*
        mult*          subst*         more-ideal*
        div*           diff*          clear-ideal*
        power*         equal*         ideal*

 The following are functions changing the form of an expression.

             to-r-poly*          to-e-poly*
             to-r-rational*      to-e-rational*
             to-i*               to-p*

 The parameters show you the state of calculation.

        current-ideal$      ideal-element$
        base$               ideal-list$

If you want to know these functions, try "describe*".
When you  finish calculation, try "end*".
"help*" will help you.


Please input function or parameter.
=)set-ideal*

Please input the name of ideal.
=)ideal1

Please input the element of ideal.
=)((x * y + x ^ 2) (x * y ^ 2 + 2))
==)IDEAL1


Please input function or parameter.
=)current-ideal$
IDEAL1
```

```
Please input function or parameter.
=)ideal-list$
(SYSTEM-IDEAL$ IDEAL1)


Please input function or parameter.
=)base$
((X 3 1 0 2) (Y 1 1 0 (X 1 1)))


Please input function or parameter.
=)ideal-element$
((X * Y + X ^ 2) (X * Y ^ 2 + 2))


Please input function or parameter.
=)add*

Please input the object of calculation.
=)(((x + y) ^ 7) (r * y ^ 2 - 5) 4)
==)(R * X ^ 2 - 1)


Please input function or parameter.
=)describe*

Please input function name.
=)add*


            ADD* (f1 f2 f3 ...)

 The function "add*" adds all elements of the given list.

      ADD* (f1 f2 f3 ...) ==) f1 + f2 + f3 + ...


[ example ]     Please input function or parameter.
                =)add*
                Please input the object of calculation.
                =)(x (2 * x ^ 3 - 5 * x) 3 5)

                ==)(2 * X ^ 3 - 4 * X + 8)

 If you have made ideal and still in the state, formulas are
 manipulated on its quotient ring.



Please input function or parameter.
=)end*
END
```