

国産数式処理システム GALにおける内部表現

佐々木達昭, 元吉文男
(理研) (電総研)

INTERNAL REPRESENTATIONS OF FORMULAS IN JAPANESE COMPUTER ALGEBRA SYSTEM GAL

Tateaki Sasaki*) and Fumio Motoyoshi**)

*) The Institute of Physical and Chemical Research
Wako-shi, Saitama 351, Japan

***) Electrotechnical Laboratory, Ministry of ITI
Niihari-gun, Ibaraki 305, Japan

ABSTRACT

This paper describes internal representations of formulas in a computer algebra system GAL which we are developing. In particular, the emphasis is laid on the usefulness of expression tags, description of data structures of some common expressions, and explanation of a unique method of controlling arithmetic.

§1. Introduction

The GAL is a Lisp-based computer algebra system which we are now developing with the following two main objectives: The first is to provide a general algebra system for mathematicians to implement various algorithms easily, and the second is to provide a powerful and useful computer system for algebraic problem solving in various application areas.

The first objective requires GAL to be a general system containing many different kinds of data types. On the other hand, a high speed is an important condition in attaining the second objective. These two objectives are superficially conflicting because the computation speed is normally decreased if we make the system general by adding many data types. In order to solve this dilemma, we introduced expression tags making the type check quite efficient. As far as the present authors know, no existing algebra system uses expression tags, although the type tag is quite familiar in many Lisp systems. We found that the expression tag is also useful in increasing the modularity and extensibility of the system.

The expression tags are used only for so-called canonicals in GAL. Another important data structure for representing the formula is so-called prefix-form. The data structure of the prefix-form in GAL is rather unique, as we will see in §4. In §5, we explain a strategy for controlling arithmetic in GAL, which is quite unique and which we believe to be quite useful for the user. Finally, §6 presents empirical data indicating the speed of GAL.

§2. Canonicals and expression tags

The representations of formulas in GAL are classified into three classes, Basic-NUMbers, CANONicalS, and PREFIX-Forms, where capital letters denote the abbreviated names. The ENF definition of the BNUM is

```
<BNUM> ::= <INTEger> | <RATional-Number> |  
          <FLOATing-number> | <INTerVal> .
```

We assume that these numbers are contained in the host Lisp system. (Currently, not every Lisp system contains RATNUM data type, but RATNUM will be available soon in most Lisp systems. The INTVL is available only in a few Lisp systems now, but the authors recommend INTVL as a useful data type in Lisp systems for scientific use.) The INT, RATNUM, and FLOAT should be arbitrary precisions, of course.

By canonical representations in computer algebra, we mean some prescribed and unique data structures representing mathematical expressions. Most computer algebra systems define canonical representations for common mathematical expressions such as polynomials, although the representations are considerably different from system to system. There are several important merits of doing so. First, mathematically equivalent expressions are uniquely represented, making the equivalence check quite easy. Second, a high computation speed is attained. Third, fixing the data structure is very convenient for implementing many algebraic algorithms. In addition to these merits, it is desirable to define the data structures for structured mathematical expressions such as matrix or set. In the GAL, canonical data structures are defined for ALGebraic-Numbers, POLYNomials, RATional-Functions, ALGebraic-Functions, SERIES,

VECTors, MATrices, TENSors, SETs, and Ordered-SETs. (Currently, not all of these are implemented.) Furthermore, GRAPHS etc. are planned to be incorporated into GAL as canonicals.

In some systems, different canonicals are distinguished from each other by the difference of data structures. Since the GAL contains so many canonicals, this method is useless. In order to distinguish many canonicals clearly and rapidly, we put an expression tag at the head of each list representing a canonical. The expression tag is a Lisp short integer (normally 24 bits length), and it is composed of three parts, global type tag, local type tag, and domain-property tag. For the significance and usage of these tags, see [1]. In this paper, we want to emphasize the usefulness of expression tags for table-driven procedure call.

We explain this point further by an example. In the current GAL, the global type tags of POLY and RATF are 2 and 3, respectively. The multiplication procedure for POLY and RATF is stored in TABLE*CANOCANO(2,3). Hence, given two CANOs as multiplicands, we first check their global type tags and take out the necessary procedure from TABLE*CANOCANO. The multiplication procedures for POLY and RATF, for example, are divided further into detailed procedures according to the domain-property tags. (For POLYs and RATFs, domain-property tags represent types of their numeric coefficients.)

With the above-mentioned scheme of procedure call, we can program the procedures for one CANO almost independently of other CANOs. Hence, as for CANOs, GAL is highly modular and extensible.

§3. Data structures of common canonicals

In the following, by "bare-CANO" we mean a canonical data structure with the expression tag deleted. By "A . B" and "(E₁ ... E_n)" we mean the dotted pair of A and B and the list with elements E₁, ..., E_n, respectively. We use VAR and TRAN as abbreviations of VARIABLE and TRANSCENDENTAL-Number, respectively.

The BNF definition of POLY data structure is as follows:

```
<bare-POLY> ::= (<main-VAR> <P-C pair> ... <P-C pair> ; *)
<main-VAR> ::= <commutative VAR> | <TRAN> ;
<P-C pair> ::= <POWER> . <COEFF> ;
<POWER> ::= <nonnegative INT> ;
<COEFF> ::= <BNUM> | <ALGN> | <bare-POLY> ;
```

*) The main-VAR is the highest order VAR in the bare-POLY, and the <P-C pair>s are in decreasing order.

For example, when the VAR x is of higher order than the VAR y, the canonical representation of $x^2 + 2xy + 1$ is

```
(#TAG x (2 . 1) (1 . (y (1 . 2))) (0 . 1)).
```

This data structure is, when the tag is deleted, quite similar to the polynomial representation in MACSYMA [2], although not exactly the same. It should be noted that a TRAN is treated as a VAR in the internal representation. However, a TRAN is constructed to be of lower order than any VAR, hence the TRAN appears in coefficients of VARs. It should be noted further that functions such as sin(x) or cos(y) are not allowed as variables of polynomials.

Using POLY data structure, RATF data structure is defined as

```
<bare-RATF> ::= (<D-N pair> ... <D-N pair> ; *)
<D-N pair> ::= <DENOM> . <NUMER> ;
<DENOM> ::= <bare-POLY> | <BNUM> | <ALGN> ;
<NUMER> ::= <bare-POLY> | <BNUM> | <ALGN> ;
```

*) The <D-N pair>s are in increasing order.

The pair of each denominator and numerator is determined by preserving the input form, unless the user specifies the collection of fractions. However, the fractions of the same denominator are collected into a single fraction. Therefore, the RATF data structure is not canonical in a strict sense, because there are many different data structures which are mathematically equivalent. For example, $1/(xy+2x+y+2) - 1/(x+1) - 1/(y+2)$, which is equal to 0, is represented as

```
(#TAG (y+2 . -1) (x+1 . -1) (xy+2x+y+2 . 1)).
```

where underlines denote bare-POLY data structures.

The POLY data structure is used also in ALGN and ALGF:

```
<bare-ALGN> ::= <bare-ALGF> ::= <NUMER> . <DENOM> ;
<NUMER> ::= <BNUM> | <bare-POLY> ; *)
<DENOM> ::= <BNUM> | <bare-POLY> ; *)
```

*) The VARs in <bare-POLY> are algebraic identifiers.

We can always normalize the DENOM to 1, but the normalization is often time consuming. Hence, we allow unnormalized representations. In this sense, the representations of ALGN and ALGF are not truly canonical.

In determining the data structures of POLY and RATF, we considered the high performance to be the most important condition. In fact, if the RATF is represented by a single fraction, which is canonical when the

common factor in the denominator and numerator is cancelled, the representation will be large in general. Hence, our representation of RATEF will lead to a much higher performance than that in REDUCE [3], for example. On the other hand, elegance of mathematical treatment is the leading condition in determining the data structures of ALGN and ALGF. For example, the imaginary number i is represented in the following complicated form (#I is the GAL name of i)

$$(\#TAG (\#I (1 . 1)) . 1).$$

However, representing i in this way, we can treat the i just the same as other algebraic numbers. (If a user wants to manipulate i efficiently, he can introduce a variable, say I , with the simplification rule $I**2 \rightarrow -1$.)

§4. Data structures of prefix-forms

Although the canonical representations are quite useful, we must introduce also prefix-forms into GAL. There are mainly two reasons for doing so. The first is that canonical representations are not able to represent various forms of expressions which are often necessary in actual calculations. For example, we often need a factored form $(x + 1)^2(x + 2)^2$ or $(x^2 + 3x + 2)^2$ instead of $x^4 + 6x^3 + 13x^2 + 12x + 4$. The second reason is that the existence of canonical form is unknown for many mathematical expressions. Furthermore, if we define canonical representations for most expressions in GAL, much memory will be expended for the tables such as TABLE*CANOCANO.

The first element of the list representing the PREF is an identifier.

Hence, the PREF can be distinguished clearly and easily from the CANOS which have integers as the first elements. All the prefix-forms in GAL are classified as follows:

$$\langle \text{PREF} \rangle ::= \langle \text{Rational-EXPRession} \rangle \mid \langle \text{SET-EXPRession} \rangle \mid$$

$$\langle \text{RELational-EXPRession} \rangle \mid \langle \text{LOGical-EXPRession} \rangle .$$

The first elements of the lists representing SETEXPR, RELEXP, and LOGEXPR are, respectively, set theoretic operators \cup , \cap , etc., relational operators $=$, $>$, etc., and logical operators \vee , \wedge , etc. The RATEXP is constructed in the following way:

- (1) It may contain BNUMs as numbers;
- (2) It may contain VARS (commutative or noncommutative) and TRANS;
- (3) It may contain unstructured CANOs (hence, ALGN, POLY, RATEF, ALGF, and SERI);
- (4) It may contain mathematical FUNCTIONS (commutative or noncommutative);
- (5) It may contain OPERATORS;
- (6) Operation of addition, multiplication, division, exponentiation, and composition are allowed in any nested way.

Although the RATEXP covers very wide classes of expressions, they are classified into only three forms, C*-form, NC*-form, and C+-form. The C*-form and NC*-form represent products of factors which are mutually commutative and noncommutative, respectively. The C+-form represents the sum of BNUM, CANOs, NC*-forms, C*-forms, and C+-forms. We give the BNF definitions of these forms.

$$\langle \text{C+-form} \rangle ::= (\text{C+} \langle \text{BNUM} \rangle \langle \text{CANO} \rangle \dots \langle \text{CANO} \rangle$$

```

<T-C pair> ... <T-C pair> ; *1)
<T-C pair> ::= <prefix-TERM> . <numeric-COEF> ;
<prefix-TERM> ::= <NC*-form> | <C*-form> |
<commutative C+-form> ; *2)
<numeric-COEF> ::= <BNUM> ;
<C*-form> ::= (C* <CBASE> <POWER> ... <CBASE> <POWER>) ; *3)
<NC*-form> ::= (NC* <NCBASE> <POWER> ... <NCBASE> <POWER>) ;
<CBASE> ::= <BNUM> | <unstructured CANO> |
<commutative FUNC> | <NC*-form> |
<commutative C+-form> ; *4)
<NCBASE> ::= <noncommutative VAR> | <OPER> |
<noncommutative FUNC> |
<NC*-form> | <noncommutative C+-form> ;
<POWER> ::= <BNUM> | <unstructured CANO> | <RATEXPR> ;
<FUNC> ::= <FUNC-name> . (<ARG> ... <ARG>) ;
<ARG> ::= nil | <BNUM> | <CANO> | <RATEXPR> ;

```

- *1) The <BNUM> and <CANO>s may be nil, <CANO>s are in increasing order, and <T-C pair>s are in decreasing order;
- *2) Ordering of constructors is NC* > C* > C+;
- *3) The <CBASE>s are in decreasing order;
- *4) The number of <NC*-form> is 1 at most.

In most systems, "+", "-", "*", "/", and "**" are used as constructors of prefix-forms. In our representations, three of these are eliminated by introducing simple structures into prefix-forms. This elimination allows us to arrange the terms in C+-forms or factors in C*-forms in

quite a natural order. Furthermore, arithmetic on RATEXPRs becomes quite simple by this elimination. Note that the C+-form can contain C*-forms and NC*-forms simultaneously, which is necessary because we often handle sums of noncommutative and commutative terms. Since the constructor NC* is of higher order than C* and C+, and since the C+-form does not contain noncommutative C+-form as a term, NC*-forms are arranged near the head of the C+-form. Hence, we can easily check whether or not a given C+-form contains noncommutative elements.

The followings are some examples of RATEXPR representations, where underlines denote CANO representations, "sin" is assumed to be of high order than "cos", and "A" and "B" are noncommutative VARs:

$$\begin{aligned}
 \sin(x)^2 \cos(x) & \implies (C* \sin x)^2 (\cos x) 1 ; \\
 (2x^2+1)(x+1)^2 & \implies (C* \underline{(2x^2+1)} 1 \underline{(x+1)} 2) ; \\
 (x+1)^3 A \cdot B & \implies (C* (NC* A 1 B 1) 1 \underline{(x+1)} 3) ; \\
 4 \sin(x)^3 + 3 \cos(x)^4 / (x+2) + (x^2+x+1) & \implies \\
 (C+ \underline{(x^2+x+1)} (C* (\sin x)^3) \cdot 4) & ((C* (\cos x)^4 \underline{(x+2)} -1) \cdot 3) .
 \end{aligned}$$

It should be commented that using canonical representations for the arguments of FUNC is not a waste of memory, because the polynomial data structure for a single variable is made unique as far as possible.

§5. Arithmetic control

For controlling arithmetic, such as whether or not a factored form is expanded, GAL provides a useful and practical method. The principle for arithmetic control in GAL is structure preserving. In fact, the repre-

sentation of RATF has been determined on this principle. This principle leads us to the following unique method of controlling arithmetic.

For the addition operation, GAL allows three modes, UNCOLLECTION, COLLECTION, and STRONG-COLLECTION. Let u and v be nonzero BNUM, CANO, or RATEXPR. In UNCOL mode, addition of u and v results normally in a C+-form unless both u and v are BNUMs. For example, if u and v are CANOs such that u is of higher order than v , we have

$$u + v \implies (C+ v u).$$

In the COL mode (default), any BNUMs and CANOs are unified, and any C+-forms are unified to a single C+-form unless the result reduces to a simpler form, where CANOs in C+-forms are unified only when they are of the same type. Important is the addition of C*-forms. Two C*-forms are unified only when they differ from each other by a CANO factor of power 1. For example,

$$\begin{aligned} \sin(x)^3(x+1)(x-1) + 2\sin(x)^3(x-1)(y+2) \\ \implies \sin(x)^3(x+2y+5)(x-1), \end{aligned}$$

but $\sin(x)^3(x+1)^2(x-1)$ and $\sin(x)^3(x-1)^2(y+2)$ are not unified. On the other hand, in STCOL mode, any BNUMs and CANOs in C+-forms are unified, and any C*-forms are unified when they differ from each other by CANO/BNUM factors of integer powers. For example,

$$\begin{aligned} \sin(x)^3(x+1)^2(x-1) + \sin(x)^3(x-1)^2(y+2) \\ \implies \sin(x)^3(x^3 + x^2y + 3x^2 - 2xy - 5x + y + 1). \end{aligned}$$

For the multiplication and exponentiation operations, GAL allows also three modes, UNEXPANSION, EXPANSION, and STRONG-EXPANSION. Let u and v be nonzero BNUM, CANO, or RATEXPR. In UNEXP mode, exponentiation of u

results normally in a C*-form and multiplication of u and v does also unless both u and v are BNUMs. For example, if u and v are CANOs such that u is of higher order than v , we have

$$u \times v \implies (C* u v 1).$$

In the EXP mode (default), CANOs and BNUMs are multiplied as usual and C+-forms are expanded, but any CANOs and C+-forms in C*-forms are not expanded. For example,

$$\begin{aligned} \sin(x)^2(x+1)^2(z+3) \times \cos(y)^3(x+1)(y+2) \\ \implies \sin(x)^2\cos(y)^3(x+1)^3(y+2)(z+3). \end{aligned}$$

On the other hand, in STEXP mode, all the BNUMs, CANOs, and C+-forms of integer powers are expanded.

Thus, once factored forms or power forms are constructed, their structures are preserved until the user issues COLLECT/EXPAND command or adds/multiplies them in STCOL or STEXP modes. (The COLLECT command and STCOL mode do not always destroy the structure.) Note that, the arithmetic in COL and EXP modes is almost the same as that in our paper-and-pencil calculations. One practical problem in calculating large expressions is how to avoid the expression swell, and a human solves this problem by structure preserving. Our method of arithmetic control seems to be a simple and useful, although not sufficient, solution to this problem.

§6. Performance

We compared GAL with REDUCE-2 in the speed of basic arithmetic. The following timing data were obtained using CLISP system [4] on FACOM/M-380

computer. (The REDUCE-2/CLISP on M-380 is the current fastest version of REDUCE. The REDUCE-3 is about 1.15 times slower than REDUCE-2.) In the following table, expressions S, A, B, C, and D are

$$S = 2x^{37} + 3x^{33} + 4x^{27} + 5x^{21} - 6x^{13} - 7x^9 - 8x^5 - 9x^3 - 10$$

$$A = x^2y + xy^2 + y^2z^2 + z + 1, \quad B = A + yz,$$

$$C = (x + y + u)(x + z - v) + x + u, \quad D = A + y - v,$$

$$E = \sin(x)(x+y) + \cos(y)(z+1) + (x-y), \quad F = E + (z+1).$$

good, although the superiority to REDUCE is lost for polynomials of many variables. It is remarkable that arithmetic on prefix-forms is quite efficient in GAL. Usually, performance of arithmetic on prefix-forms is considerably worse than that on canonicals.

Acknowledgement. We thank Prof. S. Watanabe, Dr. Y. Kanada, and Mr. H. Murao for discussions on the representations of polynomials.

unit/msec	GAL	REDUCE-2
S**7	211**)	3,582**)
A7 = A**7	65	182
B7 = B**7	126	340
A7 * B7	4,292	21,791
A7 / A**5	61**1)	211**2)
B7 / B**2	151	506
C4 = C**4	103	166
D4 = D**4	162	219
C4 * D4	9,045	19,194
E6 = E**6	103**3)	126
F6 = F**6	278**3)	336
E6 * F6	10,038**3)	20,995
E6 * F6	3,487**4)	*****

**) the result is not printed out.

*1) using DIVIDE command.

*2) using REMAINDER command on REDUCE-3.

*3) in STEXP mode.

*4) in EXP mode.

Polynomials A, B, C, D are represented in canonical forms, and expressions E and F are represented in prefix-forms in GAL. On the other hand, E and F are represented in canonical forms by treating sin(x) and cos(y) as variables in REDUCE. We see that performance of GAL is pretty

References

- [1] T. Sasaki and A. Furukawa, "Design of a general computer algebra system," IPSJ Meeting Report on Symbol Manipulation", No.23-2, March, 1983.
- [2] A. C. Hearn, "REDUCE user's manual", Version 3.0, The Rand Corporation, 1983.
- [3] The MATHLAB Group, "MACSYMA Reference manual", 9th Version, Laboratory for Computer Science, MIT, 1977.
- [4] A. C. Norman, "Introduction to the Cambridge Lisp System", Computer Laboratory, Univ. Cambridge, 1981.