記 号 処 理 31－8
（1985. 3. 12）

国産数式処理システムGALにおける制御機構と簡単化

CONTROL MECHANISM AND SIMPLIFICATION 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

The GAL is a general computer algebra system which we are constructing now. The structure and the control mechanism of GAL were designed on a simple conceptual model of formula manipulation; here, by control we mean both system control and computation control. This paper describes the control mechanism and the simplification in GAL, including the symbol table and basic methods of pattern matching.

§1. Conceptual model of formula manipulation in GAL

In designing a large and complicated system, it is rather common to set up a conceptual model of processing which the system aims at perfoming. Without such a model, the complication of the system will confuse the designer very much. In this sense, a conceptual model of formula manipulation is quite desirable for constructing computer algebra systems, but this point has been misprized so far except in a few systems such as SCRATCHPAD [3]. In the case of GAL [4], we set up a simple model and found that the model is useful also for the user for understanding the system simply. Understanding the system is necessary for deriving the system effectively.

The whole processing in GAL is divided into five phases: input, symbol evaluation, computation (in a narrow sense), simpification, and output. These five phases constitute a cycle of processing and they are executed sequentially (some of them may be skipped), as shown by Fig.1. Actual processing is a repetition of such cycles.

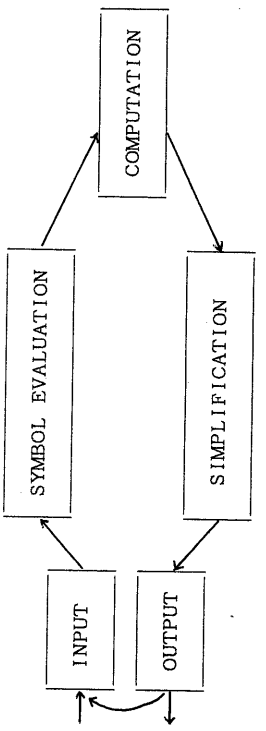

Fig.1. Cycle of five processing phases.

Let us briefly explain the phases of symbol evaluation, computation, and simplification by the following example:

```
1: RULE sin(@X)**2 -> 1 - cos(@X)**2 ;
2: A := (sin(x) + cos(x))**2 ;
3: x := s + t ;
4: t := 3 ;
5: A ;
```

Example 1. The first line defines a left-to-right rewrite rule, where the variable preceded by the character "@" is a pattern variable which is able to match with any expression. The ":=" denotes the assignment.

Note that, in the GAL, the expression is assigned to a "variable" and there is no expression number. The reason is that this assignment method is available in both interactive and batch uses. When the symbol "@X" is input in the first line, it is interpreted as independent pattern variable and converted to the polynomial representation. We call this process the symbol evaluation. Similarly, the symbols "sin" and "cos" are interpreted as function names for constructing internal representations of functions. Furthermore, in the fifth line, the symbol "A" is replaced by its value $2\sin(x)\cos(x) + 1$ and this value may be updated further because $x = s+t = s+3$. This is also the symbol evaluation. When the second line is input, computation of $(\sin(x) + \cos(x))^{**}2$ is done after the symbol evaluation phase, i.e., the addition of $\sin(x)$ and $\cos(x)$ and the exponentiation of the result. The computation phase is activated by computation activators such as +, **, GCD, or FACTOR. After the computation phase in the second line, the system applies the rewrite rule to the expression under consideration giving the result $2\sin(x)\cos(x) + 1$, which is the simplification phase.

§2. User control of symbol evaluation and simplification

We consider that each of the five phases defined in §1 is associated with an "environment" (see [5]); the result of the processing is the same if the corresponding environment is the same, while the result is usually different in different environments. The environments for input, output, and computation phases take quite many states, and the user can control most of these states by ON-OFF switches. For example, with the command "ON GCD", a rational function under computation is reduced to a simpler one by cancelling the GCD of the numerator and denominator. On the other hand, the environment for symbol evaluation has only three states, EVAL (default), UNEVAL, and STEVAL (abbreviation of strong evaluation). Similarly, the environment for simplification has three states, SIMP (default), UNSIMP, and STSIMP. (Note the naming similar to COL, UNCOL, and STCOL for the term collection mode and EXP, UNEXP, and STEXP for the term expansion mode. See [6].)

In the UNEVAL environment, the result of symbol evaluation is the symbol itself (precisely, the internal representation of the symbol). That is, even if the symbol has a value, it is not replaced by the value. Hence, if the eval environment for A is STEVAL and that for x is UNEVAL in Example 1, the result of the evaluation of A in the fifth line is

$$2\sin(x)\cos(x) + 1.$$

In the EVAL environment, the symbol is replaced by its value without updating. In the STEVAL environment, the value is updated as far as possible. For example, if the eval environment for x is EVAL and that for A is STEVAL in Example 1, the result of the fifth line is

$2\sin(s+t)\cos(s+t) + 1$,

and, if the environment for x is also STEVAL, the result is

$2\sin(s+3)\cos(s+3) + 1$.

Similarly, in the UNSIMP environment, the expression under consideration is not simplified. In the SIMP environment, each rewrite rule is applied only once to the expression. The rewrite rules are applied as far as possible in the STSIMP environment. For example, consider the Example 1 by assuming the eval environment to be STEVAL. If the simp environment for A is UNSIMP, the result of the fifth line is

$\sin(s+3)^2 + 2\sin(s+3)\cos(s+3) + \cos(s+3)^2$.

Note that the user can control the environments for function names also. For example, if the simp environments for "sin" and "cos" are UNSIMP then sin(0) and cos(0) are unchanged, while these expressions are usually simplified to 0 and 1, respectively.

The eval and simp environments can be set not only globally but also for individual symbols as the above exampes suggest. In order to set the environments globally we can use ON-OFF command, such as

ON STEVAL; OFF STSIMP;.

When some environment is switched off, the default environment is recovered. Thus, "OFF STSIMP" is equivalent to "ON SIMP". In order to set the environment for individual symbols we can use DECLARE command. For example, the command

DECL STEVAL X1,X2,X3, STSIMP X1,X2,X3;

sets STEVAL and STSIMP environments for symbols X1, X2, and X3. The individual declaration is stronger than the global declaration, unless the individual environment is default one.

In addition to the environment control mentioned above, the user can use the commands EVAL and SIMP; EVAL(X) evaluates X strongly and SIMP(X) simplifies X strongly, where X is either a symbol or an expression. Therefore, the user of GAL can control the eval and simp environments most flexibly compared with other algebra systems, but the control method is quite simple and suited for both batch and interactive uses.

§3. Simplification and pattern matching in GAL

The concept of simplification in formula manipulation is not settled yet, and the term "simplification" is used in a variety of meanings. Some people consider the simplification to be the elimination of redundant expressions such as 0*A or A+0, hence it is trivial; some people consider that the transformation of the input expression is a simplification, hence it has very wide meaning including the concepts of symbol evaluation and computation in GAL. As mentioned in §1, simplification in GAL means the formula transformation by rewrite rules. This simplification is very important in actual formula manipulation, and its importance will increase much more when the formula database will be constructed.

The simplification in the sense of GAL has been investigated and implemented by many people, in particular, in the context of term rewriting system. However, we still discuss the simplification because 1) the GAL is scheduled to equip a formula database and most mathematical formulas will be treated as rewrite rules and 2) processing a variety of

formulas efficiently requires further investigation of simplification.

The rewrite rules in GAL are classified in many different ways:

a) system built-in rules versus user defined rules;

b) open rules versus packed rules;

c) single-term rules versus multi-term rules;

d) rules with and without conditions;

and so on.

Examples of system built-in rules are derivative rules for functions and defining polynomials for algebraic numbers, while most mathematical formulas are treated as user defined. By open rules, we mean the rewrite rules which are applied at every phase of simplification. Thus, open rules are applied to every expression computed. On the other hand, some rewrite rules may be packed into a rule box and they are applied only when the rule box is called. The REDUCE [1] allows only open rules, while only packed rules are available in MACSYMA [2]. If the number of terms in the l.h.s. of rewrite rule is one then the rule is single-term rule, and others are multi-term rules. The syntax of a rule is one of the following two forms:

RULE left -> right ;,

RULE left -> right WHERE conditions ;.

Here, left is either a unstructured CANOnical or a RATional-EXPRession which may contain pattern variables. (For CANO and RATEXPR, see [4,5]). The right is either a Basic-NUMber, unstructured CANO, RATEXPR, or LOGical-EXPRession. The conditions is a condition or a sequence of conditions on pattern variables in the rule.

The system built-in rules are stored in the symbol table which will be described in the next paragraph. The single-function rewriting rules are also stored in the symbol tables even if they are defined by the user. Other rules defined by the user are stored in either open rule box or individual rule boxes, depending on the user's specification.

As for implementing pattern matching in GAL, two points should be mentioned. One is the matching of multi-term rules, and the other is the coexistence of two data structures, CANOs and PREfix-Forms (see [5]).

Let us consider, for example, the multi-term rule

sin(@X)**2 + cos(@X)**2 -> 1.

If we perform the literal matching of the l.h.s. of this rule with

$3\sin^3(x) + 2\sin(x)\cos^2(x) - 2\sin(x)$,

we see that the matching fails. However, the rule should simplify the first and second terms of this expression. In order to achieve such a simplification, GAL multiplies a pattern variable @EXTRA to the above rule and changes it to

sin(@X)**2*@EXTRA + cos(@X)**2*@EXTRA -> @EXTRA.

Then, GAL performs the matching as follows:

Matching of $3\sin^3(x)$ and $\sin^2(@X)$ @EXTRA:

@X = x, @EXTRA = $3\sin(x)$, where factor 3 is temporal.

Matching of $2\sin(x)\cos^2(x)$ and $\cos^2(@X)$ @EXTRA:

the factor 3 in @EXTRA is modified to 2.

Hence, the original expression is simplified as

$\Longrightarrow 1\sin^3(x) + 2\sin(x)(\sin^2(x)+\cos^2(x)) - 2\sin(x)$

$\Longrightarrow \sin^3(x) + 2\sin(x) - 2\sin(x) \Longrightarrow \sin^3(x)$.

Note that if we perform the simplification by modifying the rule as sin(@X)**2 -> 1 - cos(@X)**2 then we obtain a different result.

Let us explain the second point by an example:

RULE X**@N -> 0 WHERE @N > 5;.

The GAL represents X**@N in this rule in a PREF because the exponent @N is an indeterminate. However, X**@N must match with X^6, X^7, ... which are represented in CANOs. In other words, two expressions must match with each other if they are semantically equivalent. The GAL performs this semantical matching by decomposing the given expressions into constituents and comparing the constituents pairwise. That is, an expression is first decomposed into LTERM and RTERM:

<expr> ::= <LTERM> + <RTERM>;

<LTERM> is the first term of <expr>.

Next, a term is decomposed into LFACTOR and RFACTOR:

<term> ::= <LFACTOR> * <RFACTOR>;

<LFACTOR> is the first factor of <term>.

Finally, a factor is decomposed into BASE and EXPONENT:

<factor> ::= <BASE> ** <EXPONENT>.

§4. Symbol table in GAL

As in many other large systems, the fundamental of system control in GAL is a symbol table. Some reader may think that the GAL needs not prepare its own symbol table but may utilize the symbol table of host Lisp system. However, the symbol table of Lisp system is too simple to use as symbol table of large computer algebra system.

In the GAL, each nonnumeric symbol appearing in the internal representation of expression is given a symbol table which is represented by a Lisp array. The symbol table contains seven items in the current GAL, as shown by Fig.2.

1	TYPES (type1 ... typen)
2	ORDERING-#
3	(VALUE . (3-env-#))
4	(saved-VALUE . EVALenv-#)
5	(EVALflag . SIMPflag)
6	(built-in simp-RULES)
7	(print-NAME/PROCEDURE)

Fig.2. Structure of symbol table in GAL.

Note that several types in general are given to a single symbol. Every symbol in GAL is given one of the classification types, ARRAY, CONStructor, NUMber, VARiable, FUNCtion, and SYMBOL, and it is further given one of the individual types, ALGebraic-Number, TRAnscendental-Number, etc. Furthermore, some symbols may be given property types, such as UNCOMmutative. Therefore, the list in item 1 in Fig.2 contains several types in general. Every nonnumeric symbol appearing in the GAL expression is given a unique ordering number, which is the item 2. The ordering is set as

CONS > FUNC > VAR,SYMBOL > TRAN > ALGN.

The user can change the symbol ordering any times except for and ALGN and

ALGF. The three environment numbers in the item 3 are for evaluation, simplification, and ordering, and they control these environments efficiently. For the item 2 ~ 4, see [5]. The flags in the item 5 specify the eval and simp environments for individual symbols. The simp-rules in item 6 are derivative rules, inverse functions, numerical evaluation procedures, minimal polynomials for ALGN and ALGF, rules defined by RULEBOX command, etc. The item 7 saves the print name of the symbol or the procedure name for printing mathematical expression.

References

[1] A. C. Hearn, "REDUCE user's manual," Version 3.0, The Rand Corporation, 1983.

[2] The MATHLAB Group, "MACSYMA reference manual," 9th Version, Laboratory for Computer Science, MIT, 1977.

[3] "SCRATCHPAD user's manual," Math. Sci. Dept., IBM Thomas J. Watson Research Center, 1975.

[4] T. Sasaki and A. Furukawa, "Design of a general computer algebra system," IPSJ Working Group Report, WGSYM #23-2, 1983.

[5] T. Sasaki and F. Motoyoshi, "Environment problems in formula manipulation," IPSJ Working Group Report, WGSYM #27-1, 1984.

[6] T. Sasaki and F. Motoyoshi, "Internal representations of formulas in Japanese computer algebra system GAL," IPSJ Working Group Report, WGSYM #29-5, 1984.