数理解析研究所講究録
第 663 巻 1988 年 3-22

Formula Database in Computer Algebra System GAL

佐々木建昭
T. Sasaki[1], 増永良文
Y. Masunaga[2], 三枝義典
Y. Saigusa[2]
阿部昭博
A. Abe[2], 元吉文男
F. Motoyoshi[3], 佐々木睦子
M. Sasaki[1]

1) The Institute of Physical and Chemical Research
Wako-shi, Saitama 351-01, Japan

2) University of Library and Information Science
Tsukuba-shi, Ibaraki 305, Japan

3) Electro-Technical Laboratory, MITI
Tsukuba-shi, Ibaraki 305, Japan

Abstract

With the purpose of utilizing a database by an algebra system automatically, we designed a database system of mathematical formulas and implemented it on Japanese algebra system GAL. How to index the formula is crucially important for the automatic utilization of database, and we have devised three types of indices. It is shown by examples that we can make a detailed retrieval with reasonably low noise by combined use of these indices as well as GAL's pattern matcher. Furthermore, a user retrieve statement is defined which simulates the human retrieval of formula books.

§1. Introduction

Although heuristic method was widely used in the early ages of computer algebra, algorithmic method is dominantly used in the current algebraic computation. The progress of algorithm study is quite rapid, and we have now many powerful algorithms, in particular, for polynomial operations such as GCD, factorization, ideal membership, etc. However, there are still many mathematical operations for which we have only very ineffective algorithms or have no systematic algorithm yet. For example, current integration algorithms are almost powerless for many special functions (some special functions can be integrated algorithmically).

Even for such algorithmically hard operations, humans are performing calculations without any special difficulty, and the key is the utilization of formula books. This method of computation, i.e., formula-based computation is indispensable in the actual computation, and it is realized in the existing algebra systems as capability of user-definition of rewriting rules (for example, LET statement in REDUCE[1)] and RULE command in MACSYMA[2)]). However, the realization is very much restricted compared with human utilization of formula books. In order to utilize mathematical formulas by algebra systems in such a way that human utilizes, construction of formula database is absolutely necessary. Furthermore, the database must be such that not only human but also the algebra system can retrieve formulas. Among the current algebra systems, only SMP[3)] is equipped with a formula database, but it is quite unsatisfactory from the viewpoint of automatic retrieval by the algebra system.

Equipping with a formula database as a basic facility of an algebra system will surely lead to a new development of computer algebra. We think the formula-based computation is quite fruitful for the following operations.

(a) Calculating integrals for which algorithmic methods are powerless.

(b) Solving differential equations by pattern matching.

(c) Manipulation of various finite/infinite power series.

(d) Automatic simplification of wide classes of expressions.

Of course, many more researches are necessary for realizing these operations on computers, but the construction of formula database is obviously the first step of such researches. This paper describes a formula database which has been constructed as a basic unit of Japanese algebra system GAL[4].

§2. Basic considerations

In this section, we consider how the mathematical formulas are used in various calculations and investigate the necessary conditions for formula database which is retrieved automatically by algebra systems.

First of all, we note some features of mathematical formulas.

1) The number of formulas is not many compared with the data in large-scale database.
2) Renewal of data is seldom necessary (renewal is necessary only when the input data contain errors or new formulas are added).
3) Most formulas are given no names, and retrieving formulas by only their names or main function names is almost useless.

The features 1) and 2) mean that we need not worry about many typical problems for large-scale database, such as fast access to data in disk or renewal of data without causing mutual inconsistency. On the other hand, 3) means that we must devise a useful and effective indexing method for formulas.

We usually retrieve formulas from formula books by noticing not only names of operations and functions but also the structure of expressions, and there are many conventions in expressing mathematical formulas. Therefore, the ready-made database systems which are based on indexing by keywords will not be useful for our purpose. We think the indexing method is crucially important in constructing formula database. As we see below, the indexing is closely related with how to formulate the expression transformation.

Most formulas in formula books can be classified into the following two kinds.

(1) Formulas of the first kind are to perform definite operations such as integration,

summation, etc. An example of formula is $\int \exp(-x^2)dx = \mathrm{erf}(x)$.

(2) <u>Formulas of the second kind</u> are used to the expression transformation (in a narrow sense). An example of formula is $\sin^2(x) + \cos^2(x) = 1$.

Although there may be various formulas, most formulas used by algebra systems will be classified into either of the above two. Therefore, in designing a formula database for algebra system, we confined ourselves to handle the above two kinds of formulas.

The first kind formula has the following form

(F1) Op(F,...) = [right],

where "Op" is an operation name such as "Integ" or "Sum" (operator for integration and summation, respectively) and "F" is the main argument. In the GAL, "Integ" or "Sum" etc. are classified into "pseudo-functions" and clearly distinguished from ordinary functions such as "exp" or "sin". Usage of the first kind formula (F1) is very simple: only to perform the operation "Op" for the expression "F". In the retrieval of the first kind formula, we usually notice the pattern of "F" as well as the name "Op" but do not mind the [right]. Therefore, we impose the requirement (I): <u>the formula database should be such that the first kind formula (F1) is retrieved by the name "Op" and the pattern of "F"</u>.

The retrieval and usage of the second kind formula is not simple because of the following two reasons. The first reason is that the mathematical expression can be transformed in quite various ways and we must select formulas to attain a desired transformation; the second reason is that, although the adequateness of formula selection is judged by the form of transformed expression, we can specify the form only vaguely before the transformation. Consider, for example, the formula

(F2) $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \sin(\beta)\cos(\alpha)$.

Some users want to utilize (F2) as the rewriting rule

(F3) $\sin(\alpha+\beta) \rightarrow \sin(\alpha)\cos(\beta) + \sin(\beta)\cos(\alpha)$,

and the other users will utilize it as the rewriting rule

(F4) $\sin(\alpha)\cos(\beta) + \sin(\beta)\cos(\alpha) \rightarrow \sin(\alpha+\beta)$, while formula (F2) in its form is scarcely used. We must select one of (F3) and (F4) to attain the desired transformation. Therefore, we impose the requirement (II): <u>the formula database should be such that the second kind formula "[expr1] = [expr2]" can be retrieved as rewriting rule "[expr1] → [expr2]" in some cases and rewriting rule "[expr2] → [expr1]" in other cases</u>.

That the transformed expression can be specified only vaguely is crucial in devising the indexing method for formulas. One may regard the expression transformation as a procedure of finding a "transformation path" between a given expression and a space of expressions specified vaguely. Then, we must construct a rough structure of transformed expression, which seem to be not easy. On the other hand, when transforming expressions, we seldom image the transformed expressions definitely but notice only the direction of transformation. The direction of transformation by the rule "[left] → [right]" can be specified clearly by the relative difference between [left] and [right]. With this model of expression transformation, it seems to be easy to select desired formulas automatically. Therefore, we impose the requirement (III): <u>the formula database should be such that the second kind formula can be retrieved by specifying the relative difference between [left] and [right]</u>.

Although the mathematical formulas are not so many compared with the data in large-scale database, the number is considerably large. For example, formulas of Fourier transforms or elliptic functions are collected to thick books. Hence, it is desirable that the formulas for the same operation, such as integration or Fourier transformation, are grouped and saved into the database separately from other formulas. By this, we can make a fast access to data by loading only the necessary data on the main memory. Some kinds of special functions are frequently used in some application areas. Hence, it is desirable that such special functions are grouped and added to the database independently from other formulas. By this, we can construct a special-purpose database peculiar to an application calculation. Therefore, we impose the requirement (IV): <u>the formula database should be divided into sub-databases each</u>

of which is independent from others.

In the actual application of a formula to a given expression, the expression sin(x), for example, in the formula must match with sin(a), sin[cos(x)], etc. in the given expression. Processing of this complication is a job of pattern matcher of the algebra system, and it is irrelevant to the database system.

It is quite common in actual retrieval that the user gets many more formulas than he wants. The formulas which are undesirably retrieved are called "noise". We define the noise-ratio of each retrieval as

$$\text{noise-ratio} = \frac{\text{number of formulas retrieved}}{\text{number of formulas user wants}} - 1.$$

Although the noise-ratio depends on the retrieve condition as well as the user, it will be a rough measure of goodness of indexing method.

§3. Indexing of mathematical formulas

The mathematical formulas in our database are given three types of indices, which we call indices of types I, II, and III. We describe indices of types I and II in this section, and index of type III is explained in §5. In determining the indexing methods for types I and II, the following three points are regarded as important.

(a) The requirements (I), (II), (III) given in §2 are satisfied.

(b) The noise-ratio is made as low as possible.

(c) The retrieve keys can be extracted automatically.

In the following, we regard the mathematical formula "[expr1] = [expr2]" as a rewriting rule "[left] → [right]", where [left] may be either [expr1] or [expr2] so that the requirement (II) given in §2 is satisfied.

Definition 1 [keyword]. The keywords in an expression are operation names (such as "Integ" or "Sum") and function names (such as "sin" or "exp") contained. When the expression contains no operation or function but contains transcendental constants such as π or e, we use the constants as keywords. //

Definition 2 [level of keyword]. Let Key be a keyword in a given expression. The level of Key is 1 if it appears at the top level position of the expression (i.e., not in the arguments of keywords). Let Key be a keyword of level ℓ, then the keywords appearing at the top level position of the arguments of Key are of level $\ell+1$. //

Indexing of type I

(1) Let the rewriting rule be "[left] → [right]". Let [left] = Op(F,...) if the rule is of the first kind, and let [left] = F if the rule is of the second kind. Let $(f_1,...,f_m)$ be the keywords of level 1 in F, $(g_1,...,g_n)$ be the keywords of level 2 in F, and so on.

(2) We define the type I index of the rewriting rule to be

$$((f_1,...,f_m), (g_1,...,g_n), \cdots). \quad //$$

Example 1.1. $\mathrm{Integ}(\exp(-x^2),x) \rightarrow \mathrm{erf}(x)$.

The top level keyword "Integ" is an operation name, so we index the formula by the keywords in the main argumant, $\exp(-x^2)$. Hence the index is ((exp)). //

Example 1.2. $2\,\mathrm{atan}(\mathrm{sqrt}\left(\frac{x-y}{x+y}\right)\tan(\theta/2)) \rightarrow \mathrm{acos}\left(\frac{y+x\cos\theta}{x+y\cos\theta}\right)$

where $|y| \leqq |x|$ and $0 < \theta < \pi$.

The "atan" is a keyword of level 1, and "sqrt" and "tan" are of level 2. Hence the index is ((atan) (sqrt tan)). //

The index of type I is very simple, yet it gives more information than a simple set of keywords. However, it does not satisfy the requirement (III) given in §2. Hence, we determine the indexing of type II as follows.

Indexing of type II

(1) For a given expression F, let $(K_1,...,K_m)$ be the keywords of level 1 in F. We characterize F by the number of terms, the term-degree, and the degree of each keyword K_i. We abbreviate these quantities to #TERM, #DEG, and $\deg(K_i)$, respectively. (The counting rule for these quantities are given below.)

(2) Let the rewriting rule be "[left] → [right]". If the rule is of the first kind, with [left] = Op(F,...), then we define the type II index to be the characteristic

numbers for F, otherwise we define the index to be the increase-decrease relations of characteristic numbers for [left] and [right]. //

Counting rule for #TERM, #DEG, and the keyword degree.

For a monomial $v_1^{d_1} \cdots v_m^{d_m} f_1^{e_1} \cdots f_n^{e_n}$, with $d_i > 0$ and $e_j > 0$, where $v_1, \ldots, v_m$ are variables and $f_1, \ldots, f_n$ are functions, we count as $\#DEG = \Sigma_{i=1}^{m} d_i + \Sigma_{j=1}^{n} e_j$ and $\deg(f_j) = e_j$. Below, the keyword degree is counted similarly as #DEG. For the expression $T_1 + \cdots + T_n$, where each T_i is a monomial, we count as #TERM = n and $\#DEG = \max\{\#DEG(T_i) \mid i=1,\ldots,n\}$. For expressions $F^{n/2}$ and $\mathrm{sqrt}(F)^n$, we count as #TERM = #TERM(F) and #DEG = #DEG(F)×(n/2). For the expression N/D, where D is not a number, we count as #TERM = #TERM(D) and #DEG = −#DEG(D). //

Example 2.1. $\mathrm{Integ}(\exp(-x^2),x) \to \mathrm{erf}(x)$.

The main argument of [left], $\exp(-x^2)$, is characterized as

(#TERM=1, #DEG=1, deg(exp)=1).

Hence, the index is (#TERM=1, #DEG=1, exp=1). //

Example 2.2. $\sin(\alpha+\beta) \to \sin(\alpha)\cos(\beta) + \sin(\beta)\cos(\alpha)$.

The [left] and [right] are characterized, respectively, as

(#TERM=1, #DEG=1, deg(sin)=1, deg(cos)=0),

(#TERM=2, #DEG=2, deg(sin)=1, deg(cos)=1).

Therefore, the index is (#TERM=up, #DEG=up, sin=same, cos=up). //

Note 1. The operation name "Op" of the rule "Op(F,...) → [right]" does not appear in the indices, and it is used as the sub-database name (see §5).

Note 2. For some expressions, we cannot count #TERM and/or #DEG definitely. An example is $x + x^2 + \cdots + x^n$. In such cases, we give "INDEF" as the key status.

Note 3. Formulas which are equivalent but represented differently may be indexed differently in the above scheme. For example, $(x+1)^{1/2}$ contains no keyword but sqrt(x+1) contains the keyword "sqrt". Solving this complication of indexing is not easy. One reasonable solution is to standardize the source data of formulas by a preprocessor.

In the retrieval of a formula "Op(F,...) → [right]", we can specify F by both indices of types I and II. Similarly, in the retrieval of the second kind formula, both indices of types I and II are combinedly used. Therefore, we expect the retrieval of low noise-ratio, which is checked in §6. The above indices are enough simple for computer to extract retrieve keys automatically.

§4. User retrieve statement

In our formula database, not only the GAL system but also the user can retrieve formulas by using a retrieve statement. From the user's viewpoint, usefulness of the database depends not only on the indexing but also on the retrieve statement. We have defined the statement so as to satisfy the following requirements.

(a) The retrieve statement is as simple as possible.

(b) Detailed retrieval is possible by the statement.

(c) The user can retrieve database as if he retrieves formula books.

Before defining the statement, let us consider how we retrieve formulas from formula books. The typical retrieve process will be as follows.

Typical process of human retrieval of formula books

Step 1: Regarding the desired formula as a rewriting rule "[left] → [right]", we image the structure of [left] (the structure will often be definite for the first kind formula, while it will often be vague for the second kind formula);

Step 2: Using keywords in [left], we open a relevant section of the book;

Step 3: Scanning formulas in the section, we retrieve formulas the left hand sides of which match with [left];

Step 4: Among formulas retrieved, we select some of them by checking the relation between the left and right hand sides of each formula (this step will be unnecessary in the retrieval of the first kind formula).

In some cases, we perform the following retrieval.

Retrieval by name: Retrieval by formula names, names of persons (such as Gauss or

Newton), or by names of operation or subjects. //

We have determined the retrieve statement so that it may simulate the above process naturally as follows:

```
FIND <pattern expression>
   WITH <pattern&retrieve condition>
      ABOUT <list of names>;
```

(Note 1) <pattern expression> may be "@" (see below).

(Note 2) WITH- and/or ABOUT-clause may be null.

(Note 3) "WITH" may be written as "WHERE".

Let us explain this statement in detail.

The <pattern expression> is the same as ordinary expression except that it may contain pattern variables; the pattern variable is a variable which matches with any expression, and it is a symbol beginning with character "@" in GAL. When the <pattern expression> is not specified, the FIND statement is written as "FIND @ WITH ···". The usage of pattern expression is simple and any user handling mathematical expressions frequently will be able to use the pattern expression easily. For example,

(P1) (sin(@X)**@N + cos(@X)**@N)*@Y

is a pattern expression which matches with any of the following expressions:

(E1) $e^x[\sin^3(x) + \cos^3(x)]$ · · · · @X=x, @N=3, @Y=e^x

(E2) $\sin(x+y) + \cos(x+y)$ · · · · @X=x+y, @N=1, @Y=1

(E3) $\dfrac{\sin(z)+\cos(z)}{\sin(z)-\cos(z)}$ · · · · · · · · @X=z, @N=1, @Y=$\dfrac{1}{\sin(z)-\cos(z)}$

It is needless to say that the pattern expression is suitable for representing rough structures of expressions.

The <pattern&retrieve condition> is a combination of <pattern condition> and <retrieve condition>, and it is separated to each condition by the GAL parser. The pattern expression in GAL may be conditioned by WITH-clause as constraints on pattern variables or pattern sub-expressions. For example, if the above pattern (P1) is conditioned as "WITH DENOM(@Y) = 1" ("DENOM" is a selector of denominator), then the pattern matches with (E1) and (E2) but not (E3). Next, the <retrieve condition> specifies the formula index of type II in a simple way. For example, when performing the retrieval of the first kind formula, the <retrieve condition> will be like

WITH #DEG = 3 AND 2 <= #TERM <= 4 AND sin = 2.

For specifying <retrieve condition> for the second kind formula, we use words "up", "down", "same", and "none" as follows. Let Key denote a keyword, #TERM, or #DEG, and let $d = |\text{Key-value for [right]}| - |\text{Key-value for [left]}|$. We write up(Key,...), down(Key,...), same(Key,...), and none(Key,...) for representing the cases $d>0$, $d<0$, $d=0$, and non-existence of Key in [right], respectively. For example,

WITH down(sin, cos) AND NOT up(#TERM, #DEG).

The ABOUT-clause has the following two meanings.

(1) Our formula database is divided into sub-databases (see §5 for details). By specifying the sub-database name, we can narrow the range of retrieval.

(2) We can retrieve formulas by names of formulas, persons, etc.

As an example of (1), consider the function name "sin". This keyword appears mainly in formulas of trigonometric functions, but we can also find "sin" in the integration formulas, etc. Therefore, if we do not specify the sub-database name, the retrieval is made over many sub-databases.

Let us check how the FIND statement simulates the human retrieval mentioned above. The structure of [left] of the formula is expressed by <pattern expression>. We can represent both clear and vague expressions by pattern expressions. A sub-database can be viewed as a section of a formula book. By specifying a sub-database name (or names), the user can "open a required section (or sections)". The <pattern expression> is also used as [left] in Step 3, and the relation between [left] and [right] in Step 4 can be specified by <retrieve condition>. Furthermore, retrieve by name can be realized by the ABOUT-clause.

The FIND statement is quite simple, yet it allows us to perform detailed retrieval so long as we specify the <pattern expression> and <pattern&retrieve condition> suitably. Therefore, we think that the FIND statement satisfies the requirements (a) ~ (c) given at the beginning of this section, except for the following problems on ABOUT-clause.

Among the names in the augument of ABOUT-clause, the names of formulas and persons are universal to the user, while the names of sub-databases etc. are not so because there are many similar names. Hence, it is unreasonable to expect the user to specify the names of sub-databases etc. correctly. Furthermore, misspelling of names will happen frequently. We will solve these problems by the following mechanisms.

(1) Word matcher which detects misspelling.

(2) Tables for associating related names, etc.

So far we have implemented only the mechanism (1). The usefulness of the above mechanisms is not clarified yet.

§5. Composition and function of formula database

Our formula database consists of a database management system and data sets. The management system is composed of two units, a sub-database builder and a formula retriever. The data sets are composed of the following four kinds of tables.

(1) Tables of the formula indices of type I.

(2) Tables of the formula indices of type II.

(3) Tables of the formula indices of type III.

(4) Tables in which actual formulas are saved.

Let us explain the fourth tables first. The formulas are classified into many classes and saved into many files so that the formulas in the same class are saved into the same file and formulas in different classes into different files. Each file is called a sub-database and given a unique name. In each sub-database, every formula is given a unique number and saved so that we can easily take out it by specifying the formula number. Each sub-database is completely independent from others, so that the requirement (IV) in §2 is satisfied.

The classification of formulas and the sub-database name are specified by the system builder in a systematic way as follows. First, the formulas of the form "Op(F,...) → [right]" are saved into the same sub-database of the name "Op". Second, formulas which are classified into the same class conventionally are saved into the same sub-database which is named after the class. For example, the sub-database "Trig" contains formulas for transforming trigonometric functions. Third, the sub-database name may be peculiar to users in some application areas. For example, the name may be "HEphysics" if the user constructs a sub-database containing some kinds of special functions for peculiar calculations in high energy physics.

Note. The same formula may be saved into different sub-databases. For example, some of the integration formulas of the form $\int \exp(-xt) f(x)\, dx = g(t)$ will be saved into both "Integ" and "Laplace" sub-databases.

According to the above classification of formulas, the keywords of level 1 of [left] of a formula are closely related with the corresponding sub-database name. Hence, in indexing each formula, we construct an association list (our system have been implemented in Lisp) such that

((Key1 sub-DB11 ...) (Key2 sub-DB21 ...) ···)

which means that formulas with the top level keyword "Key1" appear in sub-databases "sub-DB11", ..., and so on. This list is used to find sub-databases to be retrieved when no sub-database name is specified by ABOUT-clause.

Each formula read into the sub-database is given indices of types I and II. Thus, we obtain index lists as shown in Figure 1. Note that the index lists contain formula numbers and not the formulas themselves. By this, we can efficiently perform set operations which are necessary in the retrieve procedure. The index lists are then converted to the so-called inverted-index lists which are saved into the tables (1) and (2) mentioned above. The inverted-index list is a nested association list, as shown by Figure 2. We see that the inverted-index list is suited for getting formula numbers by specifying indices, while the index list is suited for getting indices by specifying formula numbers.

|| Fig.1 || || Fig.2 ||

When reading formulas into sub-databases, every formula with name (formula name, person name, etc.) is checked and a list

(formula number, the name, sub-database name)

is saved into another file. We call the index thus obtained the index of type III. The list of type III indices is also converted to an inverted-index list and saved into a file, which is the above-mentioned table (3). All of the above-mentioned works are done by the sub-database builder.

Let us next explain the formula retriever. Given a FIND Statement or a FIND command issued by GAL, the retriever works as follows.

System retrieve procedure

Step 1: Find the name(s) of sub-database(s) to be retrieved and load the necessary tables. Finding the sub-database name is done by checking the ABOUT-clause first, and if no name is found then the finding is done by checking the keywords of level 1 in the <pattern expression> or <retrieve condition>.

Step 2: Calculate the type I index of <pattern expression> and, if the index is not null, retrieve the index table of type I. If the <retrieve condition> is not null then retrieve the index table of type II. If both retrievals are made then take the intersection of the formula numbers retrieved. If no formula number is retrieved then goto Step 4.

Step 3: Using the formula numbers retrieved, retrieve formulas from the formula-saving table. Then, perform the pattern matching of [left] of each formula with the <pattern expression> under the <pattern condition>, and discard the formula if matching fails. Return the remaining formulas.

Step 4: If the ABOUT-clause contains names which are not sub-database names, then retrieve the index table of type III and return all the specified formulas. //

§6. Evaluation of the indexing

Let us consider the system retrieve procedure presented above. In the retrieval of the first kind formula, the <pattern expression> is usually given to specify actual expressions considerably definitely. Hence, even if many formulas are retrieved in Step 2 of the above retrieve procedure, most of them will be discarded at Step 3; the pattern matching in Step 3 is not time-consuming because both <pattern expression> and [left] of formula are small-sized. On the other hand, in the retrieval of the second kind formula, low noise-ratio is crucially important. One reason is that the <pattern expression> in this case is often given to specify actual expressions only vaguely (it will often be "@"), hence Step 3 of the above retrieve procedure is not well effective for formula selection. Another reason is that adequateness of the formula retrieved is judged only after applying to a given expression, but the expression is usually of large-sized and the application takes a lot of time.

|| Fig.3 || || Fig.4 ||

We have evaluated our indexing method by many examples. Here, we give some typical examples. Figure 3 shows some retrievals of the first kind formulas (retrieval in the sub-database "Integ"). Fig. 3-1 for the retrieval without selection by pattern matching, and Fig. 3-2 for the retrieval with selection by pattern matching. Figure 4 shows retrievals of the second kind formulas (retrieval in the sub-database "Trig"). Fig. 4-1 for the retrieval by type I index only, Fig. 4-2 for the retrieval by type II index only, and Fig. 4-3 for the retrieval by indices of types I and II. We see that, although index of type I or II alone is not so effective, combined use of them works pretty effectively. However, the noise-ratio depends on the <retrieve condition> largely, and it becomes quite small if we give a good condition (in fact, the noise-ratio in the last example of Fig. 4-3 is almost 0).

§7. Concluding remarks

In this paper, we have classified the mathematical formulas in formula books into the first and second kinds, devised three types of indexings suitable for automatic utilization of formula database, and showed usefulness of the indexings. Furthermore, we presented a simple and desirable composition of formula database, such as subdivision into sub-databases. The indexings and system composition was determined mostly from the viewpoint of practical usefulness.

Although we have tested our indexings through FIND statement, the test is not enough; in particular, it is unknown whether or not the indexings are effective for formula-based integrator or series manipulator, etc. Probably, the details of indexings will be modified when we will implement these operations actually (for example, a keyword for rational functions may be introduced). However, we think that the basic composition of our database will be unchanged.

Various notations are used in mathematics and some of them are not easy to handle by computers (for example, consider path integrals); the GAL parser is pretty general yet it cannot read many of the formulas. Constructing formula database requires us not only to input formula data and devise effective indexing but also to develop various supporting facilities such as general pattern matcher and parser, and in particular we need an algebra system which accepts various kinds of expressions. Still, the formula database on an algebra system is not truely useful if it is not utilized by an algebra system automatically. Therefore, we may say that we have passed only one step of a long way. Development of various utility modules, such as formula-based integrator or series manipulator, is now waiting for us.

Aknowledgements

This work was supported financially by The Kurata Foundations in 1982 and by The

Ministry of Education, Science and Culture (Grant-in-Aid for Scientific Research) in 1985-86 fiscal years.

References

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

2) The MATHLAB Group, "MACSYMA reference manual", Version 9, Lab. Computer Science, MIT, 1977.

3) Cole, C.A., Wolfram, S., et al., "SMP handbook", Version 1, CALTEC, 1981.

4) Sasaki, T., "Japanese algebra system GAL" (preprint in preparation).

5) Sasaki, T., "Simplification of algebraic expression by multiterm rewriting rules", Proc. SYMSAC'86, ACM, New York (1986), p. 115.

Figure captions

Fig. 1. Example of index list of type I.

Fig. 2. Example of index list of type II.

Fig. 3. Examples of retrievals of integration formulas. The results are showed by formula numbers and not the formulas themselves.

Fig. 4. Examples of retrievals of trigonometric formulas. The selection by pattern matching (Step 3 of the retrieve procedure) is skipped to see the effectiveness of indexings.

```
(350 (!#DEG . UP) (!#TERM . SAME) (SINH . SAME) (SUM . UP))
(351 (!#DEG . SAME) (!#TERM . SAME) (SINH . NONE) (SUM . UP))
(352 (!#DEG . UP) (!#TERM . SAME) (SINH . SAME) (SUM . UP))
(353 (!#DEG . SAME) (!#TERM . SAME) (SINH . NONE) (SUM . UP))
(354 (!#DEG . UP) (!#TERM . SAME) (SINH . SAME) (SUM . UP))
(355 (!#DEG . SAME) (!#TERM . SAME) (SINH . NONE) (SUM . UP))
(356 (!#DEG . UP) (!#TERM . SAME) (SINH . SAME) (SUM . UP))
(357 (!#DEG . UP) (!#TERM . UP) (COSH . UP) (SINH . SAME))
(358 (!#DEG . DOWN) (!#TERM . SAME) (COSH . NONE) (SQRT . UP))
(359 (!#DEG . UP) (!#TERM . SAME) (COSH . NONE) (SINH . UP)
     (SUM . UP))
```

Fig. 1. Example of index list of type II

```
(COMBI (UP 157 166 395 397))
(COS
   (DOWN 85 89 159 160 161 162 163 164 165 405 406 411 416 533
      535)
   (INDEF 537)
   (NONE 10 18 19 21 23 24 31 53 71 81 83 84 88 97 99 100 101
      102 103 121 128 129 130 147 148 166 167 172 173 180 181
      184 185 297 401 402 403 404 407 408 409 410 415 417 418
      420 515 529 545 551 572 573)
   (SAME 3 32 52 87 92 93 94 95 149 412 419 421 422 425 427 428
      431 534 556 557)
   (UP 8 9 13 14 15 17 20 22 25 26 33 34 35 36 37 38 39 40 50 54
      55 56 57 62 63 64 65 68 69 70 72 73 74 75 78 79 80 82 91
      104 107 108 110 112 113 114 118 119 120 122 123 124 125
      126 127 150 152 154 156 170 171 176 177 413 423 424 426
      429 430 432 522 523 527 536 554 555 560 561 564 565 567
      616 621 622 623 624 625 626 627 628 629 630 631 632 633
      634 635 636))
```

Fig. 2. Example of inverted-index list of type II

```
FIND INTEG(SINH(@X)*COSH(@X))ABOUT INTEG;
(57 58 59 60 61 65 66 82 89 90 91 92 93 94 95 96 97 100 132 133 136 137)

FIND INTEG(SINH(@X)*COSH(@X)) WITH SINH=1 AND COSH=1 ABOUT INTEG;
(57 58 59 60 61 65 66 82 93 94 100 132 133 136 137)
```

Fig. 3-1.

```
FIND INTEG(SINH(@X)*COSH(@X)) WITH SINH=1 AND COSH=1 ABOUT INTEG;
(65 66)
```

Fig. 3-2.

```
FIND SIN(@X)+SIN(@Y) ABOUT TRIG;

(1 13 14 15 16 17 33 34 35 36 37 38 39 40 50 51 54 55 62 63 68 69 78 79
82 86 90 91 96 98 104 105 106 107 108 109 110 111 112 113 114 115 116
117 144 145 146 150 151 152 153 154 155 156 157 158 168 169 182 183 514
544 548 554 555 567 569 570 571)
```

Fig. 4-1.

```
FIND @ WITH UP(#DEG,COS) ABOUT TRIG;

(8 9 13 14 15 17 20 22 25 26 34 36 38 40 54 55 56 57 62 63 64 65 68 69
70 72 73 74 75 78 79 80 104 107 108 110 112 113 114 118 119 120 122 123
124 125 126 127 170 171 176 177 423 424 426 429 430 432 522 523 527 536
537 554 555 560 561 564 565 567 621 622 623 624 625 626 627 628 631 632
633 634 635 636)

FIND @ WITH UP(#DEG,COS) AND DOWN(#TERM) ABOUT TRIG;

(68 69 70 78 79 80)
```

Fig. 4-2.

```
FIND SIN(@X)+SIN(@Y) WITH UP(#DEG,COS) ABOUT TRIG;

(13 14 15 17 34 36 38 40 54 55 62 63 68 69 78 79 104 107 108 110 112
113 114 554 555 567)

FIND SIN(@X)+SIN(@Y) WITH UP(#DEG,COS) AND DOWN(#TERM) ABOUT TRIG;

(68 69 78 79)
```

Fig. 4-3.