Derivations of Identities by Symbolic Computation

Akira Nakamura

Department of Linguistics and Information Sciences
Osaka University of Foreign Studies
Aomadani, Mino City, Osaka 562-8558
Japan
E-mail: naka_work@jttk.zaq.ne.jp

Received October 28, 2005
Revised August 4, 2006

We have made the implementation of symbolic computation programs which can derive identity relations for arbitrarily given mathematical expressions. Simulations of relatively easy several concrete examples have been shown to run within practical speeds.

Key words: identity relation, orthogonal polynomial, Toda equation

1. Introduction

The uses of rapidly developing symbolic computation software such as Mathematica [13], Maple [4] and Risa/Asir [11, 12] in the fields of sciences, technologies and educations are becoming very useful and in many cases indispensable. Naturally in applied mathematics fields this tendency is especially strong. By the present-day symbolic computation software we can perform differentiations and integrations relatively easily for the given mathematical expressions. For the illustrations of the following arguments, let us take a simple and easy example by the so-called Hermite polynomial $H_n(x)$. The definition and their low-order concrete expressions are given by [1, 5]

$$H_n(x) \equiv (-1)^n \exp\left(\frac{x^2}{2}\right)\left(\frac{d}{dx}\right)^n \exp\left(-\frac{x^2}{2}\right), \tag{1}$$

$$\begin{aligned} &H_0(x) = 1, \quad H_1(x) = x, \quad H_2(x) = x^2 - 1, \quad H_3(x) = x^3 - 3x, \\ &H_4(x) = x^4 - 6x^2 + 3, \quad H_5(x) = x^5 - 10x^3 + 15x, \ldots. \end{aligned} \tag{2}$$

Since this function is well-known, their differential equation and recurrence relations (hereafter abbreviated as r.r.) can be obtained from the standard literature as [1, 5]

$$\left[\left(\frac{d}{dx}\right)^2 - x\left(\frac{d}{dx}\right) + n\right] H_n(x) = 0, \tag{3}$$

$$H_{n+1}(x) - xH_n(x) + nH_{n-1}(x) = 0. \tag{4}$$

Several low order expressions of equation (4) can be written explicitly as

$$\begin{aligned} &H_2(x) - xH_1(x) + 1H_0(x) = 0, \quad H_3(x) - xH_2(x) + 2H_1(x) = 0, \\ &H_4(x) - xH_3(x) + 3H_2(x) = 0, \quad H_5(x) - xH_4(x) + 4H_3(x) = 0. \end{aligned} \tag{5}$$

As stated above it is easy to verify that equations (5) actually hold by the present-day symbolic computation software.

In the research activities very different situation arises. When a new set of research materials are given, we want to discover the relations so far unknown among them. We usually do trial and error searches for the unknown relationships based on one theoretical expectation or another. In these searches, it will be nice to have the help of rapidly developing power of symbolic computation software. In this paper we consider the creation of the symbolic computation programs which help above searches.

More precisely our aim can be explained in the following steps from (a) to (e).

(a) We assume that we have explicitly-known mathematical expressions given by the various forms such as the orthogonal polynomials, special functions, their determinants, pfaffians... and so forth.

(b) We want to find the new unknown identity relations such as the differential relations, r.r. and so forth satisfied by the mathematical expressions prepared in (a).

(c) In the process (b), we assume that we can obtain certain basic knowledge about special/low-order relations, but not general/higher-order relations.

(d) Based on the hints suggested by the special/low-order relations, we perform the trial and error searches for the general/higher-order relations.

(e) In order to help the searching processes in (d), we consider the creation (and the use) of the symbolic computation programs which give output of identity relations (which mathematically correspond to the ones such as differential relations, r.r. and so forth) for arbitrarily chosen input of mathematical expressions.

Now we explain by the illustrative examples. Let us assume that the quantity $H_n(x)$ can be given explicitly by the definition (1), but the differential equation such as equation (3) and r.r. such as equation (4) are unknown and we want to discover them. We like to create the program which helps this process of discovering the relations such as equations (3) and (4). Concretely, we are supposed to type the input commands such as

(comp. 1.1)

```
H=newvect(20)$  Name=newvect(20)$
for(N=0;N<15;N++){H[N]=naka_H(N,x)$  Name[N]="H("+rtostr(N)+",x)"$}
     cputime(1)$
for(N=1;N<4;N++){naka_FinderOfIdentityRelation([H[N+2],H[N+1],H[N]],
     [Name[N+2],Name[N+1],Name[N]])$}
     cputime(0)$
```

where `naka_H(N,x)` stands for the program to create Hermite polynomial and `cputime(1,0)` is the command for measuring cpu-time consumed for the calculation. To such input we want to have the computer program which returns the very direct output such as

(comp. 1.2)

```
### YES Relation Exists! ###
     (-1)*(H(3,x))+(x)*(H(2,x))+(-2)*(H(1,x))=0.
### YES Relation Exists! ###
     (-1)*(H(4,x))+(x)*(H(3,x))+(-3)*(H(2,x))=0.
### YES Relation Exists! ###
     (-1)*(H(5,x))+(x)*(H(4,x))+(-4)*(H(3,x))=0.
```

and so forth where `H(n,x)` stands for $H_n(x)$. In the later section, slightly modified commands of (comp. 1.1) written as

(comp. 1.3)

```
P1=naka_H(N+2,x)$  P2=naka_H(N+1,x)$  P3=naka_H(N,x)$
Name1="H("+rtostr(N+2)+",x)"$  Name2="H("+rtostr(N+1)+",x)"$
Name3="H("+rtostr(N)+",x)"$    cputime(1)$
naka_FinderOfIdentityRelation([P1,P2,P3],[Name1,Name2,Name3])$
cputime(0)$
```

with varying N values will be used to measure size-effect of calculation time. Throughout the paper, the term '`list`' denotes the programming data given by square bracket, comma, and elements such as `[a,b,c]`. Note that although the input material of above examples are mathematically linear expressions in $H_n(x)$, they can be chosen as the nonlinear expressions with respect to (hereafter abbreviated as w.r.t.) $H_n(x)$ as will be shown later.

In the following we will show the basic scheme and the explanations of the concrete programming and the commands. Once the basic strategy is established and the concrete implementation is made in one software, implementations in other software can be expected to be basically similar. At present we have created software for Risa/Asir and Maple. We present the concrete programming explanations by the example of Risa/Asir. (Maple case is completely parallel.) As to the naming of the commands, we follow the proposed rule [12] such that every command begins with the fixed prefix, in our case for example `naka_`, which ensures to avoid collisions with any commands already existing in the software.

2. Basic Features of the Algorithm

In this section we will consider rough overall algorithmic scheme of the calculations. For the present problem we need the following steps (I-A) to (I-F).

(I-A) Creation of command to solve the input data identically as the preparation.

Since we consider the problem of deriving the identity relations, first of all, we must tackle with the problem of identically solving an arbitrarily given expression. Needless to say, we should distinguish the difference between solving the expression and 'identically' solving the expression. For example as to the expression

$$ax^2 - (b-2)x - cx^2 = 0, \tag{6}$$

solving it w.r.t. x in the ordinary sense gives solution $x = 0, (b-2)/(a-c)$ while 'identically' solving it w.r.t. x gives different solution $a = c, b = 2$. Clearly our needs lie in the 'identically' solving the expressions. The command to be created should have input arguments consisting of input identity data and the set of independent variables data. The input data should include both independent variables and unknowns. To solve 'identically' means that the values of the unknowns included in the input should be determined such that every order of coefficients w.r.t. the independent variables should vanish.

(I-B) Gathering the independent variables from the given initial set of expressions.

Let us denote the given initial set of mathematical quantities (= input data) as $P_i = P_i(x_1, x_2, \dots)$ for $i = 1, 2, \dots, N'$ where $x_1, x_2, \dots$ are independent variables. This P_i corresponds to $H_i(x)$ of our illustrative example in the previous section. We extract all the independent variables $x_1, x_2, \dots$ from the initial data set of P_i's and provide them in a list form.

(I-C) Creation of the trial function.

Let the trial function T be the linear combination of P_i's defined by

$$T \equiv T(x_1, x_2, \dots) \equiv \sum_{i=1}^{N'} C_i^{(M_i)}(x_1, x_2, \dots) P_i(x_1, x_2, \dots). \tag{7}$$

We assume that the coefficients $C_i^{(M_i)}(x_1, x_2, \dots)$'s are the M_i-th order polynomials w.r.t. the independent variables gathered in (I-B) and also include the program-induced unknowns which we denote by the name $\mathrm{coe}_{ijk\dots}$. They can be written as

$$\begin{aligned} C_i^{(M_i)} = C_i^{(M_i)}(x_1, x_2, \dots) &\equiv \mathrm{coe}_{i00\dots} + (\mathrm{coe}_{i10\dots}\, x_1 + \mathrm{coe}_{i01\dots}\, x_2 + \cdots) + \cdots, \\ &= \sum_{a_1=0}^{M_i}{}^{\mathrm{cond}} \cdots \sum_{a_N=0}^{M_i}{}^{\mathrm{cond}} \mathrm{coe}_{i a_1 a_2 \cdots a_N} (x_1)^{a_1} (x_2)^{a_2} \cdots (x_N)^{a_N}, \end{aligned} \tag{8}$$

$$\sum{}^{\mathrm{cond}} \equiv \text{summation under condition } (a_1 + a_2 + \cdots + a_N \le M_i). \tag{9}$$

(I-D) 'Identically' solving the trial function w.r.t. the set of independent variables.

By using the command prepared in (I-A), we 'identically' solve the trial function

$$T = 0, \tag{10}$$

w.r.t. the independent variables. Namely we obtain the solutions for the program-induced unknowns $\mathrm{coe}_{i a_1 a_2 \cdots a_N}$'s.

(I-E) Simplifying all $C_i^{(M_i)}$'s by substituting the solutions of $\mathrm{coe}_{i a_1 a_2 \cdots a_N}$'s given in (I-D).

(I-F) Erasing the common multiplication factors from $C_i^{(M_i)}$'s.

From the structure of equations (7) and (10) it is obvious that if $C_i^{(M_i)}$'s are solutions then $C_i^{(M_i)}$'s multiplied by any polynomial or numeric common factors can again be the solutions and vice versa. Namely the common multiplication factors have no effect on the final results. Therefore we perform the erasing of these unnecessary common factors from $C_i^{(M_i)}$'s and obtain the simplest identity expression. Finally we display the resultant expression.

3. Details of the Algorithmic Scheme (I-A) to (I-F)

In this section we consider more details of (I-A) to (I-F).

Details of (I-A)

We will name the 'identically' solving command as `naka_SolverOfIdentity(Identity00,IndependentVariablesList00)` which determines the unknowns included in `Identity00` such that `Identity00` $= 0$ holds identically w.r.t. the independent variables given by `IndependentVariablesList00`. In order to create this command we need the following steps.

(I-A-1) We extract all the variables from the input data `Identity00`. This gives us all variables set which is the set of independent variables plus unknown variables.

(I-A-2) From above all variables set, we kick out independent variables and obtain the unknown variables set. From this we make a list of unknown variables.

(I-A-3) We extract coefficients of all orders w.r.t. all independent variables from `Identity00`.

(I-A-4) We determine the values of unknowns such that the coefficients of all orders w.r.t. independent variables obtained in (I-A-3) vanish.

Now we will check (I-A-1) to (I-A-4). The process (I-A-1) can be done by Risa/Asir command '`vars`'. (Maple command '`indets`' plays the similar role.) The process (I-A-2) can be done relatively easily for example by creating the command working such as `naka_SetSubtract([a,b,x,y],[x,y])=[a,b]`. As to (I-A-3) we note that Risa/Asir command '`coef`' picks up the coefficient of the polynomial of given single order w.r.t. single independent variable. By using this we can make command which provides coefficients of all orders w.r.t. one independent variable named as `naka_CoefficientsOfAllOrdersWithRespectToOneVariable(Identity00,OneVariable00)`. By using this command we can make the command which provides coefficients of all orders w.r.t. all independent variables named as `naka_CoefficientsOfAllOrdersWithRespectToAllVariables(Identity00,VariablesList00)`. Next we consider (I-A-4). We consider the case where unknowns can be solved in the linear framework. (Note that here solving means solving in the ordinary sense and not in the 'identical' sense.) Due to the simplicity of the linearity we can create the command which gives the values of unknowns which make the linear quantities vanish. The illustrative example of this command can be seen by `naka_SolverOfLinearMultiComponentsByMultiVariables([a*y-c-2,7*x-14],[x,y])=[[a*y-c-2,7*x-14],[0,0],[x,y],[2,(c+2)/(a)],YES_SolutionExists]`. By using these commands, it is now easy to create the command of 'identity'-solver. Its operational example can be seen by `naka_SolverOfIdentity((a-c)*x^2+(b-d)*x*y+(a-5)*y,[x,y])=[[a,b,c,d],[5,d,5,d],[x,y],YES_SolutionExists]`. Here in the output list, the first and second elements correspond respectively to the unknowns and their solutions.

Details of (I-B)

Again, Risa/Asir command 'vars' gives us all the variables of not only any given polynomial but also a set of polynomials. (Maple command 'indets' plays the similar role.) So use of this command is sufficient for (I-B).

Details of (I-C)

Now we must consider the command to create the polynomial $C_i^{(M_i)}(x_1, x_2, \dots)$ in equation (8). This should include all lower-order monomials up to M_i-th order w.r.t. the given set of independent variables and every monomial term should contain every different unknown variable. The input data to this command should contain the name of unknown, a list containing a set of independent variables and the order number M_i.

We note that although the sum in equation (8) suggests program coding of N-tuple sum loops, usually programming language does not support unconditional straight-forward loops of such kind when the value of N is not known before-hand. Thus in order to realize automatic programming rigorously, we must avoid this N-tuple sum loops. It happens that by using the scheme of p-nary number we can change the present N-tuple sum loops to the single sum loop as shown in the following. We note that the total number of terms in equation (8) is less than $(M_i + 1)^N$ since each exponent a_j at most takes the value $0, 1, \dots, M_i$ for $j = 1, \dots, N$ and some terms are to be dropped by the condition (9). Thus it is sufficient to consider single integer index, say k, which takes the values as $k = 0, 1, 2, \dots, (M_i + 1)^N$. For each values of k we can derive its $(M_i + 1)$-nary representation by the formula

$$k = i_N * (M_i+1)^{N-1} + \cdots + i_2 * (M_i+1)^1 + i_1 * (M_i+1)^0, \quad (0 \le i_1, i_2, \dots, i_N \le M_i). \tag{11}$$

We express this $(M_i + 1)$-nary number by the list

$$k = [i_1, i_2, \dots, i_N]. \tag{12}$$

If we obtain data given by equation (12), then by using it we can make unknown variable having name for example $\text{coe}_{i_1 i_2 \dots i_N}$ and monomial of the form $(x_1)^{i_1}(x_2)^{i_2} \cdots (x_N)^{i_N}$. By summing up these monomials w.r.t. the single sum index k from 0 to $(M_i + 1)^N$, (with sifting them by the condition (9)) we can create polynomial given by equation (8). By this way we can change the multiple sum loops to the single sum loop. This ensures the programming workable for the cases of fully arbitrary number of independent variables and polynomial order M_i. We have created the command as stated above. As the illustrative example, we can check that for example the coefficient $C_1^{(2)}$ can be produced by

(comp. 2)

```
naka_MakerOfPolynomialWithUnknownCoefficients("coe1",[x1,x2],2)
  =coe120*x1^2+(coe111*x2+coe110)*x1+coe102*x2^2+coe101*x2+coe100,
```

where `coe120`$, \dots ,$ `coe100` are programming variables for $\mathrm{coe}_{120}, \dots, \mathrm{coe}_{100}$. This finishes the creation of $C_i^{(M_i)}$.

Next we assume that the input data P_i's and the maximum orders M_i's in equation (7) are given respectively by the list form as `IdentityDataList00` $= [P_1, P_2, \dots, P_{N'}]$ and `MaxOrderList00`$=[M_1, M_2, \dots, M_{N'}]$. We first collect independent variables from the P_i-list and then create $C_i^{(M_i)}$ by the command in (comp. 2) and sum up the product of $C_i^{(M_i)} P_i$ and easily obtain the trial function T given by equation (7). The command working for this is named as `naka_MakerOfTrialFunction(IdentityDataList00,MaxOrderList00)`. The output of this command is chosen as output $=$ [`TrialFunction`$, [C_1, C_2, \dots, C_{N'}], [P_1, P_2, \dots, P_{N'}]]$.

Details of (I-D) and (I-E)

We simply use the command obtained in (I-A) and identically solve trial function equation (10). From the resulting output of `naka_SolverOfIdentity()`, we obtain the set of unknowns and their solutions. By substituting thus-obtained solutions of the unknowns in the quantity $[C_1, C_2, \dots, C_N]$, we can do step (I-E).

Details of (I-F)

We consider the following steps.

(I-F-1) We erase common polynomial factors from the input data set.

(I-F-2) We erase common numeric factors from the input data set.

As to (I-F-1), for a given data set, we derive the least common multiplier of the denominators and greatest common divisor of the numerators and then multiply them by the former and divide them by the latter. As to (I-F-2), for a given data set, we first pick out all orders of coefficients w.r.t. all independent variables. Then we pick out all numeric constants from them. For this set, we perform similar procedures as in (I-F-1).

By using all of above procedures we now create the command `naka_FinderOfIdentityRelation00(IdentityDataList00,NameList00,MaxOrderList00)`. By using this command we can create final commands which do final searching loop to obtain the final identity relation. As to the final searching loop we consider the two types. In the first type (= default type), we perform the loop as `MaxOrderList00`$=$ $[M_1, M_2, M_3, \dots] = [0, 0, 0, \dots], [1, 1, 1, \dots], [2, 2, 2, \dots]$ and stop the loop when identity has the solution. This default command is named as `naka_FinderOfIdentityRelation(IdentityDataList00,NameList00)`. In the second type we perform the loop of command `naka_FinderOfIdentityRelation00(IdentityDataList00,NameList00,MaxOrderList00)` as `MaxOrderList00` $=[0, 0, 0, \dots], [1, 1, 1, \dots], [2, 2, 2, \dots]$ under the extra condition such that each component of `MaxOrderList00` does not exceed the corresponding value of `MaxOrderList11` $= [M_1, M_2, M_3, \dots]$. For example when `MaxOrderList11` $= [1, 0, 3, \dots]$ the loop goes like `MaxOrderList00` $= [0,0,0,\dots],[1,0,1,\dots],[1,0,2,\dots],[1,0,3,\dots]$ and so forth and stops when identity has the solution. The command of the second type is named as `naka_FinderOfIdentityRelationWithOrderConditions(IdentityDataList00, NameList00,MaxOrderList11)`.

4. Simulations for Nonlinear Input Expressions

In this section we report the results of the simulations. The examples given in Section 1 have been obtained easily. We will consider more heavy (advanced) calculations. Recently the calculations of highly nonlinear systems attract research interests such as in the so-called nonlinear completely integrable systems [3, 10]. In the field, the various different kinds of solutions of the Toda equation have been investigated [6, 7, 8, 9]. The $2+1$-dimensional Toda equation

$$L^+L^-\log V_n(x,y)-V_{n+1}(x,y)+2V_n(x,y)-V_{n-1}(x,y)=0,\quad L^\pm\equiv\frac{\partial}{\partial x}\pm\frac{\partial}{\partial y},\tag{13}$$

can be reduced by the dependent variable transformation, $V_n(x,y)=V_n^{(0)}(x,y)+L^+L^-\log f_n(x,y)$, to the so-called bilinear Toda equation

$$f_n(L^+L^-f_n)-(L^+f_n)(L^-f_n)-V_n^{(0)\mathrm{Toda}}\big(C^{(0)\mathrm{Toda}}(n)f_{n+1}f_{n-1}-f_n^2\big)=0,\tag{14}$$

where $V_n^{(0)\mathrm{Toda}}(x,y), C^{(0)\mathrm{Toda}}(n)$ respectively represent the non-trivial simple solution of the Toda equation and the integration constant satisfying $L^+L^-\log C^{(0)\mathrm{Toda}}(n)=0$. We consider the simulations of the present program to the problem of finding the identity relations of the form (14). We assume that the quantity f_n can be given by certain matrix determinants,

$$f_n=|S_{n,ij}|_{1\le i,j\le N},\tag{15}$$

where $S_{n,ij}$ is written by some kind of special function. As suggested from equation (14) we put

$$P_1=f_n(L^+L^-f_n)-(L^+f_n)(L^-f_n),\quad P_2=f_{n+1}f_{n-1},\quad P_3=f_n^2,\tag{16}$$

and try to find identity relations among P_i's for the concrete data of f_n. First we investigate the case where $S_{n,ij}$ is given by the Hermite polynomial as $S_{n,ij}=H_{n+i+j-2}(kx)$ with k being arbitrary constant. We consider 5 by 5 matrix case in equation (15) and the search for $n=1,\dots,4$. We have performed the program

(comp. 3.1)

```
F=newvect(20)$
for(N=0;N<7;N++){ F[N]=naka_fnNxN_H(5,N,[0,1,2,3,4],k*x)$}
    cputime(1)$
for(N=1;N<=4;N++){Fx=diff(F[N],x)$  Fxx=diff(Fx,x)$
```

```
    P1=F[N]*Fxx-Fx^2$   P2=F[N+1]*F[N-1]$   P3=F[N]^2$
printList([" ### Hermite 5x5 det relation for N=",N," ===>"])$
naka_FinderOfIdentityRelation([P1,P2,P3],["P1","P2","P3"])$ }
    cputime(0)$
```

where `naka_fnNxN_H()` part stands for the program to create input of `5x5` determinant consisting of Hermite polynomials. The result was obtained as

(comp. 3.2)

```
### Hermite 5x5 det relation for N=1 ===>
### YES Relation Exists! ###
     (1)*(P1)+(-k^2)*(P2)+(k^2)*(P3)=0.
### Hermite 5x5 det relation for N=2 ===>
### YES Relation Exists! ###
     (1)*(P1)+(-2*k^2)*(P2)+(2*k^2)*(P3)=0.
### Hermite 5x5 det relation for N=3 ===>
### YES Relation Exists! ###
     (1)*(P1)+(-3*k^2)*(P2)+(3*k^2)*(P3)=0.
### Hermite 5x5 det relation for N=4 ===>
### YES Relation Exists! ###
     (1)*(P1)+(-4*k^2)*(P2)+(4*k^2)*(P3)=0.
```

This gives us the result that the identity relation

$$P_1 - nk^2(P_2 - P_3) = 0, \tag{17}$$

holds at least for $1 \leq n \leq 4$. By the simple direct calculation we can check that $V_n^{(0)\mathrm{Toda}} = nk^2$ is the solution of the Toda equation (13). Thus this gives us the explicit solutions of the Toda equation at least for $1 \leq n \leq 4$. Once we obtain such a relation then we can proceed to the next stage of research activities such as proving it analytically for all n which was done by the present author in the paper [6]. In the later section, slightly modified commands of (comp. 3.1) written as

(comp. 3.3)

```
Fm=naka_fnNxN_H(5,N-1,[0,1,2,3,4],k*x)$
F0=naka_fnNxN_H(5,N  ,[0,1,2,3,4],k*x)$
Fp=naka_fnNxN_H(5,N+1,[0,1,2,3,4],k*x)$  Fx=diff(F0,x)$
Fxx=diff(Fx,x)$  P1=F0*Fxx-Fx^2$  P2=Fp*Fm$  P3=F0^2$
printList([" ### Hermite 5x5 det relation for N=",N," ===>"])$
cputime(1)$
naka_FinderOfIdentityRelation([P1,P2,P3],["P1","P2","P3"])$
cputime(0)$
```

with varying N values will be used to measure size-effect of calculation time.

Next we consider the Hyper Geometric Function (hereafter abbreviated as HGF) with two variables $F_2(a, b, b', c, c', x, y)$ [2, 7]. We make the choice of $S_{n,ij} = F_2(a_i, n + j - b_c, b'_i, c, c'_i, x, y)$ where a_i, b_c, b'_i, c, c'_i are constants. The input data P_i's are selected as

$$\begin{aligned} &P_1 = f_n f_{n,xx} - f_{n,x}^2, \quad P_2 = f_n f_{n,xy} - f_{n,x} f_{n,y}, \quad P_3 = f_n f_{n,x}, \\ &P_4 = f_{n+1} f_{n-1}, \quad P_5 = f_n^2, \end{aligned} \tag{18}$$

where x, y in the subscripts denote partial differentiations w.r.t. x, y such as $f_{n,x} = (\partial/\partial x) f_n$. We choose 2 by 2 matrix case in equation (15) and consider the search for $n = 2, 3, 4$. We have performed the program

(comp. 4.1)

```
F=newvect(20)$
for(N=1;N<10;N=N+1){
  F[N]=naka_fnNxN_F2(Size0,N,AiList,Bc,BDiList,C,CDiList,x,y)$}
  cputime(1)$
for(N=2;N<5;N=N+1){Fx=diff(F[N],x)$ Fxx=diff(Fx,x)$ Fy=diff(F[N],y)$
  Fyy=diff(Fy,y)$  Fxy=diff(Fx,y)$  P1=F[N]*Fxx-Fx^2$
  P2=F[N]*Fxy-Fx*Fy$ P3=F[N]*Fx$ P4=F[N+1]*F[N-1]$ P5=F[N]^2$
  printList(["### HGF_F2 2x2 det relation for N=",N," ===>"])$
  naka_FinderOfIdentityRelation([P1,P2,P3,P4,P5],
       ["P1","P2","P3","P4","P5"])$}
  cputime(0)$
```

where `naka_fnNxN_F2()` part stands for the program to create input of 2 by 2 determinant consisting of HGF F_2. The result was obtained as

(comp. 4.2)

```
### HGF_F2 2x2 det relation for N=2 ===>
 ### YES Relation Exists! ###
   (x^3-x^2)*(P1)+(y*x^2)*(P2)+(x^2)*(P3)+(21)*(P4)+(-22)*(P5)=0.
### HGF_F2 2x2 det relation for N=3 ===>
 ### YES Relation Exists! ###
   (x^3-x^2)*(P1)+(y*x^2)*(P2)+(x^2)*(P3)+(12)*(P4)+(-13)*(P5)=0.
### HGF_F2 2x2 det relation for N=4 ===>
 ### YES Relation Exists! ###
   (x^3-x^2)*(P1)+(y*x^2)*(P2)+(x^2)*(P3)+(5)*(P4)+(-6)*(P5)=0.
```

(The constants values were chosen as $a_1 = -3$, $a_2 = -4$, $b_c = -6$, $b'_1 = -2$, $b'_2 = -3$, $c = 3$, $c'_1 = 11$, $c'_2 = 10$.) From these data and more simulations of similar nature, one can guess the general bilinear relations satisfied by the HGF F_2. By using thus-obtained data we can generalize the relations, give the analytic proofs and obtain so-called Casorati-type N-soliton solutions of the finite Toda equation written by the HGF F_2 as reported in the paper [7]. In the later section, modified commands of (comp. 4.1) written as

(comp. 4.3)

```
N=2$  Size0=2$  AiList=[1,2]$  C=3$  CDiList=[10,9]$
Fm=naka_fnNxN_F2(Size0,N-1,AiList,Bc,BDiList,C,CDiList,x,y)$
F0=naka_fnNxN_F2(Size0,N  ,AiList,Bc,BDiList,C,CDiList,x,y)$
Fp=naka_fnNxN_F2(Size0,N+1,AiList,Bc,BDiList,C,CDiList,x,y)$
Fx=diff(F0,x)$  Fxx=diff(Fx,x)$  Fy=diff(F0,y)$  Fyy=diff(Fy,y)$
Fxy=diff(Fx,y)$  P1=F0*Fxx-Fx^2$  P2=F0*Fxy-Fx*Fy$  P3=F0*Fx$
P4=Fm*Fp$   P5=F0^2$
printList([" ### HGF_F2 2x2 det relation for BDiList,Bc=",
BDiList,",",Bc," ====>"])$    cputime(1)$
naka_FinderOfIdentityRelation([P1,P2,P3,P4,P5],["P1","P2","P3",
"P4","P5"])$   cputime(0)$
```

with varying values of `Bc` $= b_c$, `BDiList` $= [b'_1, b'_2]$ will be used to measure size-effect of calculation time.

As to the type of the input data P_i's so far we have treated examples of polynomials. Next we consider the non-polynomial example. The Gauss HGF, $F(a, b, c, x)$, for positive integer indexes contains $\log(1-x)$ term such as $F(1, 1, 2, x) = -x^{-1}\log(1-x)$, [1, 5]. In this case the command '`vars`' works as `vars(-log(1-x)/x)` $= [x, \log(-x+1)]$, namely it has the convenient property of giving us not only x but also $\log(1-x)$ as the variables (Maple command '`indets`' also works in the parallel manner.) Thanks to this feature, our program can work also for non-polynomial input cases. We have performed the program

(comp. 5.1)

```
F=newvect(20)$  Name=newvect(20)$
for(N=1;N<=10;N++){F[N]=naka_GaussHGF(N,1,6,x)$
   Name[N]="F("+rtostr(N)+",1,6,x)"$ }
   cputime(1)$
for(N=1;N<=4;N++){naka_FinderOfIdentityRelation([F[N],F[N+1],F[N+2]],
   [Name[N],Name[N+1],Name[N+2]])$ }
   cputime(0)$
```

where `naka_GaussHGF(A,B,C,X)` stands for the program to create Gauss HGF by Risa/Asir. The result was obtained as

(comp. 5.2)

```
### YES Relation Exists! ###
 (4)*(F(1,1,6,x))+(-x-2)*(F(2,1,6,x))+(2*x-2)*(F(3,1,6,x))=0.
### YES Relation Exists! ###
 (3)*(F(2,1,6,x))+(-2*x)*(F(3,1,6,x))+(3*x-3)*(F(4,1,6,x))=0.
### YES Relation Exists! ###
```

```
 (2)*(F(3,1,6,x))+(-3*x+2)*(F(4,1,6,x))+(4*x-4)*(F(5,1,6,x))=0.
### YES Relation Exists! ###
 (1)*(F(4,1,6,x))+(-4*x+4)*(F(5,1,6,x))+(5*x-5)*(F(6,1,6,x))=0.
```

This shows that indeed the input for the present program is not limited to the pure polynomials but can contain functions like $\log(1-x)$.

From the results of the above simulations, we see that the present program can be used for the derivations of not only simple linear relations but also highly nonlinear ones.

Before concluding this section we explain additions of several commands. For people who may feel cumbersome to use names in input data, we prepared the command `naka_FinderOfIdentityRelationSimple(IdentityDataList00)` which needs only input data of identities. In the final output on the screen, instead of names like "`H(5,x), H(4,x),...`" (which was typed by user), this command gives us fixed name of "`EXPR1, EXPR2,...`." Also we have prepared two short-name commands, `naka_FindIdRel()` and `naka_FindIdRelCond()`, which are equal to the commands `naka_FinderOfIdentityRelation()` and `naka_FinderOfIdentityRelationWithOrderConditions()` respectively.

5. Calculation Speeds for Software Risa/Asir and Maple

In the above simulations we have shown that our programs can actually work for several practical calculations. It may be interesting to measure the calculation speeds of these simulations by the various symbolic computation software. For this, let us have a brief overview of the software. Mathematica [13] is perhaps the most widely used all over the world and has not only good user-interface but also many good other aspects but sometimes the calculation speed seems to be low. Maple [4] seems to have both good user-interface and relatively high speeds of calculations. Perhaps Risa/Asir might not be widely known compared to above two. Risa/Asir has not highly-developed user-interface but it is understandable because it is free software and other two are commercial ones. We will limit ourselves to pick up one free software and one commercial one. So in the following, we present the data of calculation speeds for Risa/Asir and Maple. The machine used for this is the commonly available commercial personal computer which has 3.40 GHz CPU, 1 Giga Byte RAM with Windows XP OS. The versions of software used are Risa/Asir Version 20060621 (Kobe Distribution), and Maple version 10.03. The calculation time has been measured for running the command `naka_FinderOfIdentityRelation()` part only. Namely the calculation time for the input data preparation part has been excluded from the measurement. The results are shown in the Table 1 to Table 4.

From these, we can see that Risa/Asir has always higher speeds than Maple. Risa/Asir speeds are several times or more higher than those of Maple. This kind of high speeds together with money-free easy availability (free software) seems to be the strong attractive features of the software Risa/Asir.

Table 1. The calculation time needed for the input examples of the present paper. The symbols (comp. 1.1) to (comp. 5.1) are the equation numbers. Consumed cpu-time is measured for the execution time of the command `naka_FinderOfIdentityRelation()`.

Consumed cpu-time (**in seconds**)		
input commands	Risa/Asir	Maple
(comp.1.1)	0.016	0.032
(comp.3.1)	0.16	1.5
(comp.4.1)	12	50
(comp.5.1)	0.016	0.50

Table 2. The size-effect of the cpu-time for the input by (comp. 1.3) with the various N values. Consumed cpu-time is measured for the execution time of the command `naka_FinderOfIdentityRelation()`.

Consumed cpu-time (**in seconds**)			
input commands by (comp.1.3)	maximum order of input	Risa/Asir	Maple
N=100	$O(x^{102})$	0.016	0.047
N=1000	$O(x^{1002})$	0.53	2.3
N=2000	$O(x^{2002})$	2.0	5.2
N=3000	$O(x^{3002})$	4.4	9.5
N=4000	$O(x^{4002})$	8.2	15.5

Table 3. The size-effect of the cpu-time for the input by (comp. 3.3) with the various N values. Consumed cpu-time is measured for the execution time of the command `naka_FinderOfIdentityRelation()`.

Consumed cpu-time (**in seconds**)			
input commands by (comp.3.3)	maximum order of input	Risa/Asir	Maple
N=1	$O(x^{10})$	0.016	0.08
N=10	$O(x^{100})$	0.25	1.3
N=50	$O(x^{500})$	5.3	13
N=100	$O(x^{1000})$	21	60

Table 4. The size-effect of the cpu-time for the input by (comp. 4.3) for the various BDiList and Bc values. Consumed cpu-time is measured for the execution time of the command `naka_FinderOfIdentityRelation()`.

Consumed cpu-time (**in seconds**)			
input commands by (comp.4.3)	maximum order of input	Risa/Asir	Maple
BDiList=[-1,-1],Bc=-4	$O(x^6y^4)$	0.31	4.5
BDiList=[-2,-2],Bc=-5	$O(x^{10}y^8)$	3.6	20
BDiList=[-3,-3],Bc=-6	$O(x^{14}y^{12})$	12	55

6. Conclusion

In this paper we have reported the implementations of symbolic computation programs which perform derivations of identity relations for input data of arbitrarily given mathematical expressions. We have presented the basic algorithmic schemes, actual examples of the simulations and comparisons of their calculation speeds. The present two examples of the nonlinear simulations have been picked up from the bilinear theoretical calculations of the Toda equation. These examples correspond to the reproduction of relations already known in the published papers. However the present program can be used in any new field so far as input data is given explicitly (based on the hints from the special/low-order relations) to help the trial and error type searches in the quest for the knowledge of general/higher-order identity relations. We expect that the present programs would be helpful for the higher productivity of research activities in the fields of applied mathematics and technologies.

Besides that we can also consider their possible uses for the educational purposes. Let us imagine the scene where elementary course students are first taught about the new mathematical expressions (for example the Hermite polynomials.) The very natural first curiosity would yield questions of something like "What kind of relations among neighbors (recurrence relations) do they have?" or "What kind of differential equations do they satisfy?" Obviously the present programs can be the convenient tool for them to visualize these answers by simply typing the questioning commands in the computer display. In this way we hope that the present programs might be used in the diversified purposes.

The source code of the present program will be made available also on the internet at the address `http://homewww.osaka-gaidai.ac.jp/%7Enakamura/nakamura.html`.

Appendix A. List of All Commands for Software Risa/Asir

In this appendix we have made the list of all the commands used in the present program (listed in the alphabetical order). In spite of certain writing inconveniences, we have adopted the long command names such that the names themselves can indicate the meaning of their functions as much as possible. So the explanations of the meanings of the commands are given only when they seem necessary. In the following we adopt the abbreviations such that polynomial = `poly`, vector = `vect`, integer = `int`.

1. `naka_CoefficientsOfAllOrdersWithRespectToAllVariables(Identity00,VariablesList00)`,
   type of `Identity00` = `poly`, `VariablesList00`, output = `list`.
2. `naka_CoefficientsOfAllOrdersWithRespectToOneVariable(Identity00,OneVariable00)`,
   type of `Identity00` = `poly`, `OneVariable00` = one variable, output = `list`.

3. `naka_EraserOfCommonNumericFactors(ListOrVector00)`,
   type of `ListOrVector00`, output = `list` or `vect`.
4. `naka_EraserOfCommonPolynomialFactors(ListOrVector00)`,
   type of `ListOrVector00`, output = `list` or `vect`.
5. `naka_FinderOfIdentityRelation(IdentityDataList00,NameList00)`,
   `naka_FindIdRel(same)`,
   type of `IdentityDataList00` = `list`, `NameList00` = `list` of names given in '`string`,' output = `none`.
6. `naka_FinderOfIdentityRelation00(`
   `IdentityDataList00,NameList00,MaxOrderList00)`,
   type of `IdentityDataList00` = `list`, `NameList00`= `list` of names given in '`string`,' `MaxOrderList00` = `list`, output= list consisting of coefficients C_i in vector, `IdentityDataList00`, final resulting identity expression given in '`string`' and the message of yes or no given in '`string`.'
7. `naka_FinderOfIdentityRelationSimple(IdentityDataList00)`,
   type of `IdentityDataList00` = `list`, output = `none`.
8. `naka_FinderOfIdentityRelationWithOrderConditions(`
   `IdentityDataList00,NameList00,MaxOrderList11)`,
   `naka_FindIdRelCond(same)`,
   type of `IdentityDataList00` = `list`, `NameList00`= `list` of names given in '`string`,' `MaxOrderList11` = `list`, output = `none`.
9. `naka_gcdzOfDataInListOrVector(ListOrVector00)`,
   type of `ListOrVector00` = `list` or `vect`, output = `poly`.
10. `naka_ilcmOfDataInListOrVector(ListOrVector00)`,
    type of `ListOrVector00` = `list` or `vect`, output = `int`.
11. `naka_IsMember(Object00,Variable00)`,
    type of `Object00` = `poly` or `list` or `vect`, `Variable00` = single variable, output = `true` or `false`.
12. `naka_IsMemberInPolynomial(Polynomial00,Variable00)`,
    type of `Polynomial00` = `poly`, `Variable00` = single variable, output = `true` or `false`.
13. `naka_IsZeroListOrVector(ListOrVector00)`,
    type of `ListOrVector00` = `list` or `vect`, output = `true` or `false`.
14. `naka_lcmOfDataInListOrVector(ListOrVector00)`,
    type of `ListOrVector00` = `list` or `vect`, output = `poly`.
15. `naka_MakerOfPolynomialWithUnknownCoefficients(`
    `Name00,IndependentVariablesList00,Order00)`,
    type of `Name00` = `string`, `IndependentVariablesList00` = `list`, `Order00` = `int`, output = `poly`.
16. `naka_MakerOfTrialFunction(IdentityDataList00,MaxOrderList00)`,
    type of `IdentityDataList00` = `list`, `MaxOrderList00` = `list`. The output = [`TrialFunction`, $[C_1, C_2, \ldots, C_{N'}]$, `IdentityDataList00`].

17. `naka_pNaryNumberInListForm(GivenNumber00,P00,ListLength00)`,
    type of `GivenNumber00`, `P00`, `ListLength00` = `int`, output = `list`. `GivenNumber00` = input integer, `P00` = p of p-nary number, `ListLength00` = length of the output list.
18. `naka_SetSubtract(List1,List2)`,
    type of `List1`, `List2`, output = `list`, This gives the elements of `List1` which is not contained in `List2`.
19. `naka_SetSum(List1,List2)`,
    type of `List1`, `List2`, output = `list`, This gives the elements of `List1` and `List2` with common elements counted only once.
20. `naka_SolverOfIdentity(IdentityPoly00,IndependentVariablesList00)`,
    type of `IdentityPoly00` = `poly`, `IndependentVariablesList00` = `list`, output = `list` consisting of unknown variables list, its solution, independent variables list of `IndependentVariablesList00`, string of `YesNoMessage`.
21. `naka_SolverOfLinearMultiComponentsByMultiVariables(`
    `MultiComponents00,VariablesList00)`,
    type of `MultiComponentsList` = `list` or `vector`, `VariablesList00` = `list`, output = `list` consisting of `MultiComponentsList00`, solution of `MultiComponentsList00`, `VariablesList00`, solution of `VariablesList00`.
22. `naka_SolverOfLinearMultiComponentsByOneVariable(`
    `MultiComponents00,IndependentVariable00)`,
    type of `MultiComponents00`, `IndependentVariable00` = `list` or `vect`, output = `list` consisting of `MultiComponents00`, solution of `MultiComponents00`, `IndependentVariable00`, solution of `IndependentVariable00`.
23. `naka_SolverOfLinearOneComponentByOneVariable(Comp00,Variable00)`,
    type of `Comp00`, `Variable00` = `poly` or `list` or `vect`, output = `list` consisting of `Comp00`, solution of `Comp00`, `Variable00`, solution of `Variable00`.

Appendix B. List of All Commands for Software Maple

Most of the commands for Maple have precisely equal Risa/Asir counterparts listed in the Appendix A. The word '`vector`' of Risa/Asir should be read as 1-dimensional '`array`' of Maple. Other different points for Maple are as follows.

3. `naka_EraserOfCommonNumericFactors()` is unnecessary since it is covered by the command 4.

5, 6, 8. `NameList00` = `list` of names given by Maple '`symbol`' instead of Risa/Asir '`string`.'

10. `naka_ilcmOfDataInListOrVector()` is unnecessary since it is covered by the command 14.

11. `naka_IsMember()` is unnecessary.

15. Type of `Name00` = Maple '`symbol`' instead of Risa/Asir '`string`.'

Appendix C. Full Source Code for Risa/Asir

The following is the full source code for the software Risa/Asir. Here we briefly comment about the complex input. The Risa/Asir command `red` simplifies the real expression but not complex one (containing $\sqrt{(-1)} \equiv$ `@i`). Therefore we added the command `naka_redc(Rat)` which can work for the complex input such as `naka_redc((x^2+1)/(x+@i))=x+(-1)*@i`. Due to this, the present program can also work for the complex input.

```
Naka_MaxOrderOfCoeffs=7 $

def naka_CoefficientsOfAllOrdersWithRespectToAllVariables(Identity00,
                                                           VariablesList00){
  Identity00=red(Identity00);
  if(length(VariablesList00)==0){Atai=[Identity00];}
  if(length(VariablesList00)>=1){
    CoefList=naka_CoefficientsOfAllOrdersWithRespectToOneVariable(
           Identity00,VariablesList00[0]);
    if(length(VariablesList00)==1){Atai=CoefList;}
    if(length(VariablesList00)>=2){
     List0=[];
     for(VK=1;VK<length(VariablesList00);VK++){
       for(K=0;K<length(CoefList);K++){
         CoefList22=naka_CoefficientsOfAllOrdersWithRespectToOneVariable(
         CoefList[K],VariablesList00[VK]);
         List0=append(List0,CoefList22);}}
     Atai=List0;}}
  return(Atai);}

def naka_CoefficientsOfAllOrdersWithRespectToOneVariable(Identity00,
                                                          OneVariable00){
  Identity00=red(Identity00);
  Max=deg(Identity00,OneVariable00);
  List0=[];
  for(K=Max;K>=0;K--){
    Coe=coef(Identity00,K,OneVariable00);
    if(Coe!=0){List0=append(List0,[Coe]);}}
  Atai=List0;
  return(Atai);}

def naka_EraserOfCommonNumericFactors(ListOrVector00){
  Lov=ListOrVector00;
  if(naka_IsZeroListOrVector(Lov)==true){Atai=Lov;}else{
    VariablesList=vars(Lov);
    if(VariablesList==[]){NuList=Lov;}
    if(VariablesList!=[]){Ws=[];
    for(K=0;K<length(Lov);K++){
      List00=naka_CoefficientsOfAllOrdersWithRespectToAllVariables(Lov[K],
                                                                VariablesList);
      Ws=naka_SetSum(Ws,List00);}
    NuList=Ws;}
    Deno=map(dn,NuList);
    ILCMOfDeno=naka_ilcmOfDataInListOrVector(Deno);
    if(type(Lov)==4){
      Ws=[];
      for(K=0;K<length(Lov);K++){
        XX=naka_redc(Lov[K]*ILCMOfDeno);
        Ws=append(Ws,[XX]);}
      Lov=Ws;}
    if(type(Lov)==5){
      for(K=0;K<size(Lov)[0];K++){
        Lov[K]=naka_redc(Lov[K]*ILCMOfDeno);}}
    Nume=map(nm,NuList);
    GCDOfNume=naka_gcdzOfDataInListOrVector(Nume);
    if(GCDOfNume==0){GCDOfNume=1;}
    if(type(Lov)==4){
      Ws=[];
      for(K=0;K<length(Lov);K++){
        XX=naka_redc(Lov[K]/GCDOfNume);
        Ws=append(Ws,[XX]);}
      Lov=Ws;}
    if(type(Lov)==5){
      for(K=0;K<size(Lov)[0];K++){
        Lov[K]=naka_redc(Lov[K]/GCDOfNume);}}
    Atai=Lov;}
  return(Atai);}

def naka_EraserOfCommonPolynomialFactors(ListOrVector00){
  Lov=ListOrVector00;
  if(naka_IsZeroListOrVector(Lov)==true){Atai=Lov;}else{
    Deno=map(dn,Lov);
    LCMOfDeno=naka_lcmOfDataInListOrVector(Deno);
    if(type(Lov)==4){
      Ws=[];
      for(K=0;K<length(Lov);K++){
        XX=naka_redc(Lov[K]*LCMOfDeno);
        Ws=append(Ws,[XX]);}
      Lov=Ws;}
    if(type(Lov)==5){
      for(K=0;K<size(Lov)[0];K++){
        Lov[K]=red(Lov[K]*LCMOfDeno);}}
    Nume=map(nm,Lov);
    GCDOfNume=naka_gcdzOfDataInListOrVector(Nume);
    if(type(Lov)==4){
      Ws=[];
      for(K=0;K<length(Lov);K++){
        XX=naka_redc(Lov[K]/GCDOfNume);
        Ws=append(Ws,[XX]);}
      Lov=Ws;}
    if(type(Lov)==5){
      for(K=0;K<size(Lov)[0];K++){
        Lov[K]=naka_redc(Lov[K]/GCDOfNume);}}
    Atai=Lov;}
  return(Atai);}

def naka_FinderOfIdentityRelation(IdentityDataList00,NameList00){
  extern Naka_MaxOrderOfCoeffs;
  Max=Naka_MaxOrderOfCoeffs;
  FlagOfRelationExist=0;
  for(Order00=0;Order00<=Max;Order00++){
    L0=[];
    for(K=0;K<length(IdentityDataList00);K++){
      L0=append(L0,[Order00]);}
    MaxOrderList00=L0;
    List00=naka_FinderOfIdentityRelation00(
               IdentityDataList00,NameList00,MaxOrderList00);
    if(List00[3]=="YES_SolutionExists"){
      FlagOfRelationExist=1;
      break;}}
  if(FlagOfRelationExist==0){print(" ### NO Relation Exists! ### ");}
  if(FlagOfRelationExist==1){print(" ### YES Relation Exists! ### ");
  Mess="      "+List00[2];
  print(Mess);}}

def naka_FindIdRel(IdentityDataList00,NameList00){
  nakaFinderOfIdentityRelation(IdentityDataList00,NameList00);}

def naka_FinderOfIdentityRelation00(IdentityDataList00,NameList00,MaxOrderList00){
  IDList=IdentityDataList00;
  IDListLen=length(IDList);
  ResultIdentityString="";
  IndependentVariablesList=vars(IDList);
  IVList=IndependentVariablesList;
  TrialFunList=naka_MakerOfTrialFunction(IDList,MaxOrderList00);
  Coef00=TrialFunList[1];
  SolutionList=naka_SolverOfIdentity(TrialFunList[0],IVList);
  YesNoMessage=SolutionList[3];
  if(YesNoMessage=="YES_SolutionExists" &&
       naka_IsZeroListOrVector(SolutionList[1])==false){
    for(K=0;K<IDListLen;K++){
      for(KK=0;KK<length(SolutionList[0]);KK++){
        Coef00[K]=subst(Coef00[K],SolutionList[0][KK],
                               SolutionList[1][KK]);}}
    Coef00=naka_EraserOfCommonPolynomialFactors(Coef00);
    Coef00=naka_EraserOfCommonNumericFactors(Coef00);
    Ws="";
    for(K=0;K<IDListLen;K++){
      if(K!=0){Ws=Ws+"+";}
      Ws=Ws+"("+rtostr(Coef00[K])+")*("+NameList00[K]+")";}
    Ws=Ws+"=0.";
    ResultIdentityString=Ws;
    ReturnMessage="YES_SolutionExists";}
  if(YesNoMessage=="YES_SolutionExists" &&
       naka_IsZeroListOrVector(SolutionList[1])==true){
    ReturnMessage="NO_SolutionDoesNotExist";}
  if(YesNoMessage=="NO_SolutionDoesNotExist"){
    ReturnMessage="NO_SolutionDoesNotExist";}
  Atai=[Coef00,IDList,ResultIdentityString,ReturnMessage];
  return(Atai);}

def naka_FinderOfIdentityRelationSimple(IdentityDataList){
  extern Naka_MaxOrderOfCoeffs;
  Max=Naka_MaxOrderOfCoeffs;
  FlagOfRelationExist=0;
  Ws=[];
  for(K=0;K<length(IdentityDataList);K++){
    String="EXPR"+rtostr(K+1);
    Ws=append(Ws,[String]);}
  NameList=Ws;
  naka_FinderOfIdentityRelation(IdentityDataList,NameList);}

def naka_FinderOfIdentityRelationWithOrderConditions(IdentityDataList00,
                                                     NameList00,MaxOrderList00){
  extern Naka_MaxOrderOfCoeffs;
  Max=Naka_MaxOrderOfCoeffs;
  FlagOfRelationExist=0;
  Len=length(IdentityDataList00);
  for(Order00=0;Order00 <= Max;Order00++){
    Ws=[];
    for(K=0;K<Len;K++){
      if(Order00<MaxOrderList00[K]){A00=Order00;}
      else{A00=MaxOrderList00[K];}
      Ws=append(Ws,[A00]);}
    L00=Ws;
    List00=naka_FinderOfIdentityRelation00(IdentityDataList00,NameList00,L00);
    if(List00[3]=="YES_SolutionExists"){
      FlagOfRelationExist=1;
      break;}}
  if(FlagOfRelationExist==0){print(" ### NO Relation Exists! ### ");}
  if(FlagOfRelationExist==1){print(" ### YES Relation Exists! ### ");
  Mess="      "+List00[2];print(Mess);}}

def naka_FindIdRelCond(IdentityDataList00,NameList00,MaxOrderList00){
  naka_FinderOfIdentityRelationWithOrderConditions(IdentityDataList00,
                                                    NameList00,MaxOrderList00);}

def naka_gcdzOfDataInListOrVector(ListOrVector00){
  if(type(ListOrVector00)==4){Max=length(ListOrVector00);}
  if(type(ListOrVector00)==5){Max=size(ListOrVector00)[0];}
  if(Max==1){
    X=ListOrVector00[0];
    Atai=gcdz(real(X),imag(X));}
  if(Max>=2){
    X=ListOrVector00[0];
    Atai=gcdz(real(X),imag(X));
    for(K=0;K<Max;K++){
      X=ListOrVector00[K];
      XX=gcdz(real(X),imag(X));
      Atai=gcdz(Atai,XX);}}
  return(Atai);}

def naka_ilcmOfDataInListOrVector(ListOrVector00){
  if(type(ListOrVector00)==4){Max=length(ListOrVector00);}
  if(type(ListOrVector00)==5){Max=size(ListOrVector00)[0];}
  if(Max==1){
    Atai=ListOrVector00[0];
    if(type(Atai)!=1){Atai=1;}}
  if(Max>=2){
    XX1=ListOrVector00[0];
    for(K=0;K<Max;K++){
      XX2=ListOrVector00[K];
      if(type(XX1)!=1 || type(XX2)!=1){
        Atai=1;
        break;}
      Atai=ilcm(XX1,XX2);
      XX1=Atai;}}
  return(Atai);}

def naka_IsMember(Object00,Variable00){
  Atai=false;
  Var00=Variable00;
  if(type(Variable00)==4 && length(Variable00)==1){
    Var00=Variable00[0];}
  if(type(Object00) <= 2){
    Atai=naka_IsMemberInPolynomial(Object00,Var00);}
  if(type(Object00)==4 || type(Object00)==5){
    if(type(Object00)==4){Max=length(Object00);}
    if(type(Object00)==5){Max=size(Object00)[0];}
    for(K=0;K<Max;K++){
      if(naka_IsMemberInPolynomial(Object00[K],Var00)==true){
        Atai=true;
        break;}}}
  return(Atai);}

def naka_IsMemberInPolynomial(Polynomial00,Variable00){
  Atai=false;
  if(type(Polynomial00)<2 && type(Variable00)<2){
    if(Polynomial00==Variable00){Atai=true;}}
  if(type(Variable00)==4){Var00=Variable00[0];}
  if(type(Variable00)==5){Var00=Variable00[0];}
  if(type(Variable00)==2){Var00=Variable00;}
  PVariablesList00=vars(Polynomial00);
  for(K=0;K<length(PVariablesList00);K++){
    if(PVariablesList00[K]==Var00){Atai=true;}}
  return(Atai);}

def naka_IsZeroListOrVector(ListOrVect00){
  Atai=true;
  if(type(ListOrVect00)==4){Max=length(ListOrVect00);}
  if(type(ListOrVect00)==5){Max=size(ListOrVect00)[0];}
  for(K=0;K<Max;K++){
    if(ListOrVect00[K]!=0){Atai=false;break;}}
  return(Atai);}

def naka_lcmOfDataInListOrVector(ListOrVector00){
  if(type(ListOrVector00)==4){Max=length(ListOrVector00);}
  if(type(ListOrVector00)==5){Max=size(ListOrVector00)[0];}
  if(Max==1){Atai=ListOrVector00[0];}
  if(Max>=2){Atai=ListOrVector00[0];
  for(K=0;K<Max;K++){Atai=lcm(Atai,ListOrVector00[K]);}}
  return(Atai);}

def naka_MakerOfPolynomialWithUnknownCoefficients(
      Name00,IndependentVariablesList00,Order00){
  IVList=IndependentVariablesList00;
  IVNumber=length(IVList);
  MaxTotal=(Order00+1)^IVNumber;
  Atai=0;
  for(J=0;J<MaxTotal;J++){
    L=naka_pNaryNumberInListForm(J,Order00+1,IVNumber);
    String=""; Power=0; Prod=1;
    for(K=0;K<IVNumber;K++){
      String=String+rtostr(L[K]);
      Prod=Prod*IVList[K]^L[K];
      Power=Power+L[K];}
    Term=strtov(Name00+String)*Prod;
    if(Power <=Order00){Atai=Atai+Term;}}
  return(Atai);}

def naka_MakerOfTrialFunction(IdentityDataList00,MaxOrderList00){
   IDList=IdentityDataList00;
   IDListLen=length(IDList);
   IVList=vars(IDList);
   Coef00=newvect(IDListLen);
   for(K=0;K<IDListLen;K++){
     Coef00[K]=naka_MakerOfPolynomialWithUnknownCoefficients(
                      "coe"+rtostr(K+1),IVList,MaxOrderList00[K]);}
   Sum=0;
   for(K=0;K<IDListLen;K++){Sum=Sum+Coef00[K]*IDList[K];}
   TrialFun=Sum;
   TrialFun=nm(TrialFun);
   Atai=[TrialFun,Coef00,IDList];
   return(Atai);}

def naka_pNaryNumberInListForm(GivenNumber00,P00,ListLength00){
  X=GivenNumber00;
  Atai=[];
  for(K=0;K<ListLength00;K++){
    Atai=append(Atai,[irem(X,P00)]);
    X=idiv(X,P00);}
  Atai=reverse(Atai);
  return(Atai);}

def naka_redc(Rat){
  Deno=dn(Rat);
  Nume=nm(Rat);
  DenoRe=real(Deno);
  DenoIm=imag(Deno);
  NumeRe=real(Nume);
  NumeIm=imag(Nume);
  if(DenoIm!=0 && DenoRe!=0){
    Deno2=DenoRe^2+DenoIm^2;
    Nume2Re=NumeRe*DenoRe+NumeIm*DenoIm;
    Nume2Im=NumeIm*DenoRe-NumeRe*DenoIm;
    Re=Nume2Re/Deno2;
    Im=Nume2Im/Deno2;}
  if(DenoIm!=0 && DenoRe==0){
    Re=NumeIm/DenoIm;
    Im=-NumeRe/DenoIm;}
  if(DenoIm==0 && DenoRe!=0){
    Re=NumeRe/DenoRe;
    Im=NumeIm/DenoRe;}
  AtaiIm=red(Im);
  AtaiRe=red(Re);
  Atai=AtaiRe+AtaiIm*@i;
  return(Atai);}

def naka_SetSubtract(List1,List2){
  if(List2==[]){Atai=List1;}
  else{Ws=[];
  for(K=0;K<length(List1);K++){
    Vari=List1[K];
    if(naka_IsMember(List2,Vari)==false){
      Ws=append(Ws,[Vari]);}
    Atai=Ws;}}
  return(Atai);}

def naka_SetSum(List1,List2){
  if(List2==[]){Atai=List1;}
  else{
    Ws=List1;
    for(K=0;K<length(List2);K++){
      Vari=List2[K];
      if(naka_IsMember(List1,Vari)==false){
        Ws=append(Ws,[Vari]);}
      Atai=Ws;}}
  return(Atai);}

def naka_SolverOfIdentity(IdentityPoly00,IndependentVariablesList00){
  if(type(IdentityPoly00)==2){ID=IdentityPoly00;}
  if(type(IdentityPoly00)==3){ID=nm(IdentityPoly00);}
  IVList=IndependentVariablesList00;
  MCList=naka_CoefficientsOfAllOrdersWithRespectToAllVariables(ID,IVList);
  List11=append(IVList,[ID]);
  AllVariablesList=vars(List11);
  UnknownVariablesList=naka_SetSubtract(AllVariablesList,IVList);
  UVList=UnknownVariablesList;
  List00=naka_SolverOfLinearMultiComponentsByMultiVariables(MCList,UVList);
  Atai=[List00[2],List00[3],IVList,List00[4]];
  return(Atai);}

def naka_SolverOfLinearMultiComponentsByMultiVariables(MultiComponents00,
                                                        VariablesList00){
  MC=MultiComponents00;
  MCSolution=MC;
  VList=VariablesList00;
  VListSolution=VList;
  for(K=0;K<length(VList);K++){
    List00=naka_SolverOfLinearMultiComponentsByOneVariable(MCSolution,VList[K]);
    MCSolution=List00[1];
    Ws=[];
    for(KK=0;KK<length(VList);KK++){
      XX=VListSolution[KK];
      XX=subst(XX,List00[2],List00[3]);
      XX=red(XX);
      Ws=append(Ws,[XX]);}
    VListSolution=Ws;}
  if(naka_IsZeroListOrVector(MCSolution)==true){
    YesNoMessage="YES_SolutionExists";}
  if(naka_IsZeroListOrVector(MCSolution)==false){
    YesNoMessage="NO_SolutionDoesNotExist";}
  Atai=[MC,MCSolution,VList,VListSolution,YesNoMessage];
  return(Atai);}

def naka_SolverOfLinearMultiComponentsByOneVariable(MultiComponents00,Variable00){
  MC=MultiComponents00;
  MCSolution=MC;
  IndVar=Variable00;
  IndVarSolution=IndVar;
  Flag00=0;
  for(K=0;K<length(MCSolution);K++){
    if(naka_IsMember(MCSolution[K],IndVar)==true && Flag00==0){
      IndVarSolution=naka_SolverOfLinearOneComponentByOneVariable(
                        MCSolution[K],IndVar)[3][0];
      Ws=[];
      for(KK=0;KK<length(MCSolution);KK++){
        XX=MCSolution[KK];
        XX=subst(XX,IndVar,IndVarSolution);
        XX=red(XX);
        Ws=append(Ws,[XX]);}
      MCSolution=Ws;
      Flag00=1;
      break;}}
  Atai=[MC,MCSolution,IndVar,IndVarSolution];
  return(Atai);}

def naka_SolverOfLinearOneComponentByOneVariable(Comp00,Variable00){
  if(type(Comp00)==4 && length(Comp00)==1){CC=Comp00[0];}
  if(type(Comp00)==5 && size(Comp00)[0]==1){CC=Comp00[0];}
  if(type(Comp00)==2){CC=Comp00;}
  if(type(Variable00)==4 && length(Variable00)==1){
    VV=Variable00[0];}
  if(type(Variable00)==5 && size(Variable00)[0]==1){
    VV=Variable00[0];}
  if(type(Variable00)==2){VV=Variable00;}
  if(deg(CC,VV)==1){
    VarSol=red(-coef(CC,0,VV)/coef(CC,1,VV));
    Atai=[[CC],[0],[VV],[VarSol]];}
  if(deg(CC,VV)>1){Atai=[[CC],[CC],[VV],[VV]];}
  return(Atai);}

end$
```

References

[1] M. Abramowitz and I.A. Stegun, Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. National Bureau of Standards Applied Mathematics Series, **55**, National Bureau of Standards, US Dept. of Commerce, USA, 1964.

[2] A. Erdelyi et al., Higher Transcendental Functions Vol. I. Robert E. Krieger Publishing Co., Malabar, Florida, 1953.

[3] R. Hirota, The Direct Method in Soliton Theory. Cambridge U.P., Cambridge, UK, 2004.

[4] M.B. Monagan et al., Maple Introductory Programming Guide. Waterloo Maple Inc., Canada, 2005.

[5] S. Moriguchi et al., Mathematical Formulas III. Iwanami Shoten, Tokyo, 1960 (in Japanese).

[6] A. Nakamura, Toda equation and its solutions in special functions. Journal of the Physical Society of Japan, **65** (1996), 1589–1597.

[7] A. Nakamura, Ladder operator approach of special functions and the N-soliton solutions of the $2+1$ dimensional finite Toda equation. Journal of the Physical Society of Japan, **73** (2004), 838–842.
[8] A. Nakamura, Ladder operator approach of special functions for $1+1d$ discrete systems and the N-soliton solutions of the quotient-difference equation. Journal of the Physical Society of Japan, **73** (2004), 2667–2679.
[9] A. Nakamura, Ladder operation method by dressed special functions and the Gram-type soliton solutions of the $2+1$ dimensional finite Toda equation. Journal of the Physical Society of Japan, **74** (2005), 1963–1972.
[10] Y. Nakamura, Applied Integrable Systems. Shokabo, Tokyo, 2000 (in Japanese).
[11] T. Saito, T. Takeshima and T. Hilano, Symbolic Computation Software Born in Japan. SEG Shuppan, Tokyo, 1998 (in Japanese).
[12] N. Takayama and M. Noro, Risa/Asir Drill-Mathematical Programming Introduction by using Risa/Asir. Kobe, 2002 (in Japanese).
[13] S. Wolfram, Mathematica Book. Wolfram Media, 2003.