Publ. RIMS, Kyoto Univ.
Vol. 6 (1970), 71-111

Formula Manipulations Solving Linear Ordinary Differential Equations (I)

By

Shunro WATANABE*

1. Introduction.

The linear ordinary differential equations of the type

$$(1.1)\qquad (x-a_1)^2(x-a_2)^2y''+(x-a_1)(x-a_2)(b_0+b_1x)y' + (c_0+c_1x+c_2x^2)y=0$$

or

$$(1.2)\qquad (x-a_1)^2y''+(x-a_1)(b_0+b_1x)y'+(c_0+c_1x+c_2x^2)y=0$$

can be solved, in a theoretical sense, applying the theory of Riemann's P function and of Hukuhara's confluent P function respectively ([1]).

The purpose of this paper and the series of papers to appear is to report an experiment on digital computer by formulation of the above theoretic approach for solving linear ordinary differential equations.

In this paper we study the formula manipulations of Frobenius algorithm ([1]) to test the occurence of the logarithmic terms in the solutions of a given linear ordinary differential equation at a regular singular point. If every solution at a regular singular point α has no logarithmic term, then the given differential equation can be reduced to the equation which has α as a regular point. Therefore we may ask to a computer about the possibility to reduce a given ordinary differential equation to the equation of the type (1.1) or (1.2). ([1]).

This paper consists of 2 parts, in Part I (Section 2 ~ Section 3)

Received December 11, 1969.

* Tsuda College. Formerly a member of the Research Institute for Mathematical Sciences, Kyoto University.

we describe a basic algorithm solving our problem, and in Part II (Section $4\sim 8$) we describe some programming techniques for the algorithm. In section 2, we review briefly the Frobenius' method which requires the factorization of characteristic equation of the differential equation. In section 3, we offer two algorithms for factorizing a polynomial with integer coefficients. From section 4 to section 8, the programming of the Frobenius method is explained in detail systematically introducing the list processing subroutines which could be useful for other purpose, especially for the general formula manipulation. For the purpose to describe our program, we introduce an algorithmic language L which is defined from lower level language to upper level language, language namely ALGOL 60, $L\alpha$, $L\alpha'$, and $L\beta$. Although this language is incomplete as a programming language, it is hoped at least to expose the nature of our formula manipulation to language designers.

PART I

2. Some remarks on Frobenius method for formula manipulation.

We consider a linear ordinary differential equation of the n-th order

$$(2.1)\qquad (x-\alpha)^n R_0(x) y^{(n)} + (x-\alpha)^{n-1} R_1(x) y^{(n-1)} + \cdots + R_n(x) y = 0$$

where $R_0(x), \cdots, R_n(x)$ are rational functions with rational number coefficients and are regular at rational point $x=\alpha$. α is called a regular singular point of (2.2). We assume that (2.1) has a solution of the form

$$(2.2)\qquad g(x,\lambda) = (x-\alpha)^\lambda \sum_{m=0}^{\infty} g_m (x-\alpha)^m \qquad (g_0 \neq 0).$$

For brevity we write

$$(2.3)\quad L(y) = (x-\alpha)^n R_0(x) y^{(n)} + (x-\alpha)^{n-1} R_1(x) y^{(n-1)} + \cdots + R_n(x) y,$$

so that also we have

$$(2.4)\qquad L((x-\alpha)^\lambda) = (x-\alpha) f(x,\lambda)$$

where

$$f(x,\lambda)=\sum_{k=0}^{n} R_{n-k}(x)\lambda(\lambda-1)\cdots(\lambda-k+1). \tag{2.5}$$

The function $f(x,\lambda)$, as a function of x, is regular at α, therefore

$$f(x,\lambda)=\sum_{r=0}^{\infty} f_r(\lambda)(x-\alpha)^r \tag{2.6}$$

where

$$f_r(\lambda)=\frac{1}{r!}\frac{\partial^r}{\partial x^r}f(x,\lambda)\bigg|_{x=\alpha} \tag{2.7}$$

are polynomials of λ with rational number coefficients. We substitute $y=g(x,\lambda)$ into (2.3) then we get

$$\begin{aligned} L(g(x,\lambda)) &= \sum_{m=0}^{\infty} g_m L((x-\alpha)^{\lambda+m}) = \sum_{m=0}^{\infty} g_m (x-\alpha)^{\lambda+m} f(x,\lambda+m) \\ &= (x-\alpha)^{\lambda}\sum_{m=0}^{\infty}\{g_m f_0(\lambda+m)+g_{m-1}f_1(\lambda+m-1)+\cdots+ \\ &\quad g_0 f_m(\lambda)\}(x-\alpha)^m. \end{aligned}$$

Therefore if (2.2) satisfies (2.1) the following relations must be satisfied.

$$\begin{cases} g_0 f_0(\lambda)=0 \\ g_1 f_0(\lambda+1)+g_0 f_1(\lambda)=0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ g_m f_0(\lambda+m)+g_{m-1}f_1(\lambda+m-1)+\cdots+g_0 f_m(\lambda)=0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots. \end{cases} \tag{2.8}$$

Here we construct a polynomial $F_0(\lambda)$ with integer coefficients from $f_0(\lambda)$ by multiplying the L. C. M. of all the denominators of coefficients of $f_0(\lambda)$. Then we factorize $F_0(\lambda)$ within integer coefficients as follows:

$$F_0(\lambda)=\varphi_1(\lambda)^{\nu_1}\varphi_2(\lambda)^{\nu_2}\cdots\varphi_T(\lambda)^{\nu_T}. \tag{2.9}$$

Next we classify $\varphi_1(\lambda)^{\nu_1},\cdots,\varphi_T(\lambda)^{\nu_T}$ into classes, one of them consists of

$$(\varphi_1(\lambda)^{\nu_1},\varphi_1(\lambda-k_2)^{\nu_2},\cdots,\varphi_1(\lambda-k_\tau)^{\nu_\tau}) \tag{2.10}$$

where $0<k_2<\cdots<k_\tau$, and k_i, $i=2,\cdots,\tau$, are integers. We call $F_0(\lambda)=0$ the characteristic equation of (2.1) and call roots of this equation the characteristic roots of (2.1).

An algorithm to make the classification of (2.10) is as follows.

Let

$$\varphi_i(\lambda)=a_l\lambda^l+\cdots+a_0 \tag{2.11}$$
$$\varphi_j(\lambda)=b_m\lambda_l+\cdots+b_0\,,$$

and we assume that $\varphi_i(\lambda)=\varphi_j(\lambda+k)$ $k>0$, then from

$$\varphi_j(\lambda+k)=b_m\lambda^m+(mkb_m+b_{m-1})\lambda^{m-1}+\cdots$$
$$=a_l\lambda^l+a_{l-1}\lambda^{l-1}+\cdots,$$

we get the necessary conditions

$$m=l\,,\quad a_m=b_m\,,\quad k=\frac{b_{m-1}-a_{l-1}}{m\times b_m}\,. \tag{2.12}$$

Then we can test whether $\varphi_j(\lambda+k)=\varphi_i(\lambda)$ holds or not using the k of (2.12).

Now return to the problem whether we can determine g_m so as to satisfy (2.8) or not. If we think the root of $\varphi_1(\lambda)=0$ of (2.10), as a root of $f_0(\lambda)=0$, then from (2.10) we have

$$f_0(\lambda+k_2)=f_0(\lambda+k_3)=\cdots=f_0(\lambda+k_\tau)=0\,, \tag{2.13}$$

and all the other $f_0(\lambda+k)$ are not zero, where 'is zero' means to be divisible by $\varphi_1(\lambda)$.

1) If $\nu=1$ in (2.10) then we put $g_0=C$ (any constant), and calculate g_m as follows. Moreover if the class of (2.10) consists of one element $\varphi_1(\lambda)^1$ then all g_m are determined from (2.8). If (2.10) consists of two or more elements, we must investigate as follows. Then for the λ to satisfy $\varphi_1(\lambda)=0$, $f_0(\lambda+1),\cdots,f_0(\lambda+k_2-1)$ are not zero. Therefore g_m, $m=1,2,\cdots,k_2-1$, are determined as rational functions of λ with rational number coefficients.

For $m=k_2$, we must consider the following relation

$$g_{k_2}f_0(\lambda+k_2)+g_{k_2-1}f_1(\lambda+k_2-1)+\cdots+g_0f_m(\lambda)=0 \tag{2.14}$$

where $f_0(\lambda+k_2)=0$. Therefore if the numerator of the following rational function of λ with rational number coefficients

$$g_{k_2-1}f_1(\lambda+k_2-1)+\cdots+g_0f_{k_2}(\lambda) \tag{2.15}$$

is not divisible by $\varphi_1(\lambda)$ then g_{k_2} is not determined so as to satisfy

(2.14). If the numerator of the rational function (2.15) is divisible by $\varphi_1(\lambda)$, then g_{k_2} becomes a free parameter, and for each $m=k_2+1, \cdots, k_{3-1}$, g_m has the form

$$h_{m_1}(\lambda)g_{k_2}+h_{m_2}(\lambda) \tag{2.16}$$

where $h_{m1}(\lambda)$, $h_{m2}(\lambda)$ are rational functions with rational number coefficients. In this case there need more investigations.

For $m=k_3$, we must consider following relation

$$g_{k_3}f_0(\lambda+k_3)+g_{k_3-1}f_1(\lambda+k_{k_3}-1)+\cdots+g_0f_{k_3}(\lambda)=0 \tag{2.17}$$

where $f_0(\lambda+k_3)$ is equal zero, and

$$g_{k_3-1}f_1(\lambda+k_3-1)+\cdots+g_0f_{k_3}(\lambda) \tag{2.18}$$

has the form of (2.16). If $h_{m_1}(\lambda)$ is divisible by $\varphi_1(\lambda)$, and $h_{m_2}(\lambda)$ is not divisible by $\varphi_1(\lambda)$, then g_{k_3} cannot be determined. If $h_{m_1}(\lambda)$ is not divisible by $\varphi_1(\lambda)$, then free parameter g_{k_2} is represented as

$$g_{k_2}=-\frac{h_{m_2}(\lambda)}{h_{m_1}(\lambda)}. \tag{2.19}$$

If both $h_{m_1}(\lambda)$ and $h_{m_2}(\lambda)$ are divisible by $\varphi_1(\lambda)$, then g_{k_3} becomes a free parameter. We repeat these steps up to $m=k_\tau$. Since we can determine g_m for $m=1, \cdots, k_\tau$, the solutions of (2.1) have no logarithmic term as expressed in (2.2). The number of these solutions are the numbers of free parameters remained.

2) When we cannot determine g_m for $m=1, \cdots, k_\tau$, in the case of 1) or when $\nu_1 \geqq 2$ and (2.10) has two or more elements, we put

$$g_0=C\varphi_1(\lambda)^{\xi(2)} \qquad \xi(2)=\nu_2+\cdots+\nu_\tau, \tag{2.20}$$

and when $\nu_1 \geqq 2$ and (2.10) has only one element, we put

$$g_0=C \tag{2.21}$$

where C is any constant. In these cases all the relations except the first one in (2.8) can be satisfied by sequentially determined g_m. Therefore for the case (2.20), g_m can be written as the following

$$g_m=\frac{\text{polynomial of}}{f_0(\lambda+1)f_0(\lambda+2)\cdots f_0(\lambda+m)}g_0(\lambda), \tag{2.22}$$

and this is true for the case (2.21). In these cases $f_0(\lambda)g_0(\lambda)$ contains $\varphi_1(\lambda)^{\xi(1)}$, where $\xi(1)=\nu_1+\cdots+\nu_r$.

$$L(g(x,\lambda))=f_0(\lambda)g_0(\lambda)(x-\alpha)^\lambda \tag{2.23}$$

is satisfied from the method of construction. From this we get

$$L\left(\frac{\partial^h}{\partial\lambda^h}g(x,\lambda)\right)=\frac{\partial^h}{\partial\lambda^h}(f_0(\lambda)g_0(\lambda)(x-\alpha)^\lambda). \tag{2.24}$$

Thus if $h<\xi(1)$ then

$$y=\left[\frac{\partial^h}{\partial\lambda^h}g(x,\lambda)\right]_{\lambda=\lambda_j} \tag{2.25}$$

where

$$\varphi_1(\lambda_j)=0$$

is a solution of (2.1). From (2.24), (2.25) and

$$g_0^{(h)}(\lambda_j)\begin{cases}\neq 0 & (h=\xi(2))\\ =0 & (h<\xi(2)),\end{cases} \tag{2.26}$$

we get all the solutions of (2.1) as follows

$$\begin{aligned} y=(x-\alpha)^\lambda\sum_{m=0}^{\infty}(x-\alpha)^m\{g_m^{(h)}(\lambda)+\tbinom{h}{1}g_m^{(h-1)}(\lambda)\log(x-\alpha)\\ +\cdots+g_m(\lambda)(\log(x-\alpha))^h\}\end{aligned} \tag{2.27}$$

$$h=\xi(2),\quad \xi(2)+1,\cdots,\xi(1)-1,\quad \varphi_1(\lambda)=0.$$

In case 1) no logarithmic term appears and in case 2) logarithmic term appears in the solutions of (2.1). For the solution of the form

$$y=(x-\alpha)^{\lambda-k_2}\sum_{m=0}^{\infty}(x-\alpha)^m g_m, \tag{2.28}$$

we make a classification (2.10′) instead of (2.10),

$$(\psi_1(\lambda)^{\nu_2},\ \psi_1(\lambda-k_3')^{\nu_3},\cdots,\psi_1(\lambda-k_\tau')^{\nu_\tau}) \tag{2.10′}$$

where $\psi_1(\lambda)=\varphi_1(\lambda-k_2)$, $k_3'=k_3-k_2,\cdots,k_\tau'=k_\tau-k_2$, and repeat the same algorithm.

3. Two algorithms for factorizing a polynomial with integer coefficients.

Kronecker's method ([2]) is known as a polynomial factorization algorithm, but here we offer other two methods.

1) Radix substitution method.

A polynomial factorization within integer coefficients

$$(3.1)\qquad (a_n x^n + \cdots + a_0) = (b_k x^k + \cdots + b_0)(c_l x^l + \cdots + c_0) \qquad k+l=n$$

can be analogously considered as a factorization of the number $a_n a_{n-1} \cdots a_0$ whose radix is x, also we assume a_k, b_i, c_j may take plus and minus values and

$$(3.2)\qquad |a_k|,\ |b_i|,\ |c_j| < x\,.$$

Given a polynomial

$$(3.3)\qquad f(x) = a_n x^n + \cdots + a_0\,,$$

we select a positive number M, such that

$$(3.4)\qquad |a_k|,\ |b_i|,\ |c_j| < M$$

and evaluate the number $N = a_n M^n + \cdots + a_0$, then factorize this number N into 2 numbers, say B and C. We expand B as those numbers of radix M whose coefficients take plus and minus values. This expansion is not unique. But if the degree k is fixed, then the number of possible expansions is finite, namely

$$(3.5)\qquad B = b_k^{(s)} M^k + \cdots + b_0^{(s)} \qquad s=1, \cdots, p.$$

We divide (3. 3) by

$$(3.6)\qquad b_k^{(s)} x^k + \cdots + b_0^{(s)} \qquad s=1, \cdots, p$$

and if divisible, then such (3. 6) is a factor of (3. 3), and if not divisible then other (3. 6) is checked. We repeat this process from degree $k=1$ to $k=[n/2]$, and also we repeat this for all possible choice of B.

The method of selection of M from (3. 3) is, for example, as follows. A factor polynomial $\varphi(x)$ of $f(x)$ has properties that $f(i)$ is divisible by $\varphi(i)$ for every integer i. Therefore $\varphi(x)$ coincides with one of the polynomial $\phi_l(x)$ which passes through $(n+1)$ points (i, 'any factor of $f(i)$') $i=0, 1, \cdots, n$. Therefore the maximum of absolute values of the coefficients of any factor of $f(x)$ is not greater than

$$\operatorname*{Max}_{\phi_l(x)}(\text{the maximum of absolute values of coefficients of } \phi_l(x)).$$

The polynomial $\phi_l(x)$ of degree k can be calculated using the values of $f(x)$ at points $x=0, 1, \cdots, k$, from the Newton's interpolation polynomial

$$\phi_l(x)=\mu_k x(x-1)\cdots(x-k+1)+\cdots\mu_1 x+\mu_0\,. \tag{3.7}$$

where $$\mu_i=\Delta^i f(0)/i!\,.$$

If $|f(i)|\leq m$ then $|\Delta f(0)|\leq 2m$ generally $|\Delta^i f(0)|\leq 2^i m$, consequently we get $|\mu_i|<\frac{2^i}{i!}m$. If c_i is coefficient of x^i of $\phi_l(x)$, then

$$\begin{aligned}|c_i|&\leq\mu_i+(\sum_{j_1=1}^{i}j_1)\mu_{i+1}+(\sum_{j_1\neq j_2}^{i+1}j_1\cdot j_2)\mu_{i+2}+\cdots+\mu_{i+1}(\sum_{j_l\neq j_m}^{k-1}j_1\cdots j_{k-i})\mu_k\\ &\leq m\left\{\frac{2^i}{i!}\left(1+(\sum_{j_1=1}^{i}j_1)\frac{2}{i+1}+(\sum_{j_1\neq j_2}^{i+1}j_1\cdot j_2)\frac{2^2}{(i+1)(i+2)}+\cdots\right.\right.\\ &\qquad\left.\left.+(\sum_{j_l\neq j_m}^{k-1}j_1\cdots j_{k-i})\frac{2^{k-i}}{(i+1)\cdots k}\right)\right\}.\end{aligned} \tag{3.8}$$

Taking the maximum of $\{\ \}$ for k and i satisfying $i\leq i\leq k<[\deg f(x)/2]$, we may set $M=m\times\max\{\ \}$.

2) Method of indeterminate coefficients.

From (3.1) we obtain the following relations between coefficients of the given polynomial and its factor polynomials.

$$\left\{\begin{array}{lll} a_n = & b_k c_l & (E_n)\\ a_{n-1}= & b_{k-1}c_l+b_k c_{l-1} & (E_{n-1})\\ \vdots & \qquad\cdot^{\cdot^{\cdot}} & \vdots\\ a_{l+1}= & b_1 c_l+\cdots\cdots\cdots\cdots+b_k c_{l-k+1} & (E_{l+1})\\ a_l = & b_0 c_l+b_1 c_{l-1}+\cdots\cdots\cdots\cdots+b_k c_{l-k} & (E_l)\\ \vdots & \vdots\qquad\cdot^{\cdot^{\cdot}} & \vdots\\ a_1 = & b_0 c_1+b_1 c_0 & (E_1)\\ a_0 = & b_0 c_0 & (E_0)\end{array}\right. \tag{3.9}$$

where $l\geqq k$, $n=k+l$, $a_0,\cdots,a_n$ are known and $b_0,\cdots,b_k$, $c_0,\cdots,c_l$ are unknown. Our purpose is to find all combinations of $(b_0,\cdots,b_k)$ and $(c_0,\cdots,c_l)$ which satisfies (3.9). The factor polynomial is obtained as $b_0+b_1x+\cdots+b_kx^k$, $1\leq k\leq[n/2]$.

First we put $k=1$ and factorize a_n to $b_k c_l$, factorize a_0 to $b_0 c_0$, then collect all combinations of (b_k, c_l, b_0, c_0), the number of combinations is finite. For each of these we can consider b_k, c_l, b_0, c_0 as known, therefore from $E_1, \cdots, E_{l-1}$ we can determine each of $c_1, \cdots, c_{l-1}$ as a rational function of $b_1, \cdots, b_{k-1}$ with rational number coefficients. Then substitute these $c_1, \cdots, c_{l-1}$ to $E_l, \cdots, E_n$, thus we obtain k algebraic equations of $b_1, \cdots, b_{k-1}$ with rational number coefficients. Starting from this system, eliminating variables one by one using the method of ([9]), we obtain at last an algebraic equation of higher degree of one variable, say b_1. By substituting to b_1 all integers which lie within the equation's root boundary, we obtain all integer solutions. And substituting one of these values into the equation of two variables obtained one step earlier of elimination of b_2. Continuing the same method, we obtain a family of polynomials

$$B(x)=b_k x^k+\cdots+b_0. \tag{3.10}$$

Then we try to divide $f(x)$ by $B(x)$, and check whether $B(x)$ is really a factor of $f(x)$. And then we repeat this for all possible (b_k, c_l, b_0, c_0), and again repeat the entire process increasing k up to $[n/2]$.

PART II

4. The outline.

In part II, we shall give a detail description of the program to solve our problem given in Section 2. Although this program is written in an assembler language, it is needed for our description to introduce a new algorithmic language by two reasons: (1) Since the original program had three levels, namely

the main routine to solve our problem
the polynomial and rational function manipulation subroutines
the very basic list processing subroutines,

it is desired to make clear the structure of editing of lower level routines in the sense of language design. (2) By the effect of (1), it will become easier to understand the programmed algorithm itself.

To describe the basic list processing routine, we shall define in Section 5 an algorithmic language $L\alpha$ adding the list type data structures and the simple list processing functions to ALGOL 60. For the polynomial and rational function manipulation subroutines and the main routine, it is needed to add further the functions between *structure types* to $L\alpha$, the resulting algorithmic language is called $L\alpha'$. However, $L\alpha'$ is still not powerfull enough to simplify the description of the higher level routines, we finally introduce an algorithmic language $L\beta$ in Section 7, where the conventions of mathematical notions are pursued. In Section 9, finally program of Frobenius algorithm is described.

5. $L\alpha$; List structures and operations for our purpose.

1) In the algorithms explained in section 2 and 3, we may consider integers, and rational functions as basic data, namely, as the operands of various operations. Furthermore we have treated more complex structures, for example $f_k(\lambda)$ of (2.8) can be viewed as data of the following form:

$$(5.1) \qquad (f_0(\lambda), \cdots, f_m(\lambda))$$

where $f_i(\lambda), i=0, \cdots, m$, are polynomials, and $g_k(\lambda)$ of (2.8) as

$$(5.2) \qquad (g_0(\lambda), \cdots, g_m(\lambda))$$

where $g_i(\lambda)=(h_{i0}(\lambda), \cdots, h_ik_i(\lambda))$, and h_{ij}, $j=0, \cdots, k_i$, are rational functions with rational number coefficients (see ((2·16) (2.19)). Moreover a rational function with rational number coefficients of the form

$$(5.3) \qquad \frac{n}{d}\times\frac{a_0+a_1x+\cdots+a_mx^m}{b_0+b_1x+\cdots+b_nx^n},$$

can be written as follows

$$(5.4) \qquad (x, (n, d), (a_0, a_1, \cdots, a_m), (b_0, b_1, \cdots, b_n)).$$

An example of representations of such data in computer memory is given for (5.4) as follows by Figure 1.

We call an unshaded rectangular box 'a node', and call a shaded

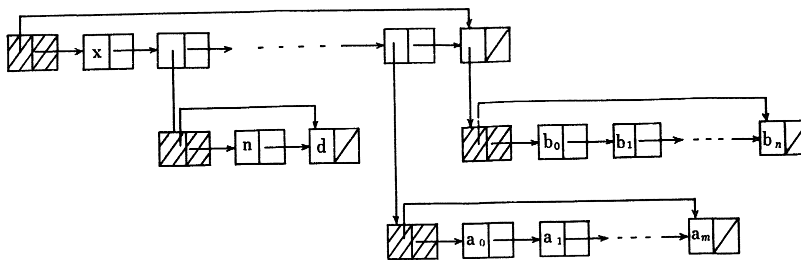


Fig. 1

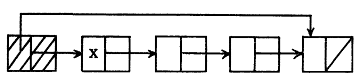


Fig. 2

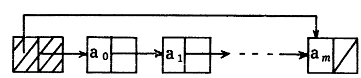


Fig. 3

rectangular box 'a head' in Figure 1. We call a sequence of nodes with a head 'a node sequence', or more precisely 'a fixed length node sequence' in the case of Figure 2, and 'a chain' in the case of Figure 3, that is of variable length.

Thus for our formula manipulations, we shall use the following definition— "a list is a node sequence or a chain and each node of which may contain an information or may refer to another head of a node sequence or a chain."

2) A head or a node consists of consecutive two words and their structures are shown in Figure 4 and 5 respectively.

DEG	EPOINTER	TYPE	POINTER

a head

Fig. 4

INFORMATION	TAG	POINTER

a node

Fig. 5

(1) EPOINTER contains an address of the right most node of the node sequence.

(2) POINTER contains an address of the node which lies just right to the head.

(3) A special address which means this node sequence ends here, is written as **nill** and this is shown by |⁄| in Figure 1.

(4) DEG (degree) contains an integer expressing (the length of the node sequence −2).

(5) TAG contains a code which distinguishes whether corresponding INFORMATION part contains an integer, a letter, or a chain.

(6) TYPE contains a code which distinguishes the type of the node sequence, and in the case of chain TYPE in partitioned to two parts, TYPE 1 and TYPE 2, these are explained later.

First we assume that all available storage locations are bound up to a chain A, the nodes of lists to be constructed are taken from this chain A. Also the heads of constructed chains and node sequences are stored in table T. In $L\alpha$ the chain A of available storage, and the area for the table T are considered to be built a priori. There are two built-in pointers pT and pA in $L\alpha$, where pT points the first unused location in T, and pA points the head of the chain A, (Figure 6 and Figure 7).

$L\alpha$ contains declarer **pointer** other than the repertories of ALGOL 60 so that a pointer variable p can be declared by '**pointer** p' where the variable p will contain an address of a head or a node.

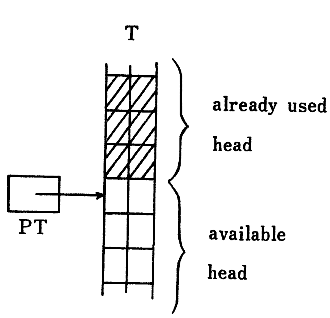


Fig. 6

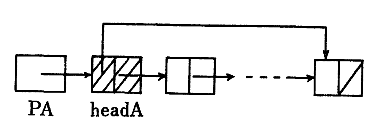


Fig. 7

When p is used in DEG (p), TYPE (p), TAG (p), EPOINTER (p), POINTER (p), and INFORMATION (p), the meaning is as follows.

If these are in the left hand side of $:=$, the value of the right hand side is stored to that field of the node or the head pointed by p, and if these are in the right hand side expression, we mean the value contained in that field.

The following is the table of basic subroutines, which can be used to construct or change lists. Pointer p_h means it points a head, and pointer p contains any node address or head address, but after head (p), p is converted to p_h. For brevity we write HEAD $(p):=$HEAD (q) this means that DEG $(p):=$DEG (q); TYPE $(p):=$TYPE (q); EPOINTER $(p):=$EPOINTER (q); POINTER $(p):=$POINTER (q);. Similarly NODE $(p):=$NODE (q) is defined.

	name	parameter	actions
(1)	head	(p)	$p:=pT$; $pT:=pT+2$; EPOINTER (p): $=$POINTER $(p):=$**nill**; DEG $(p):=-1$;
(2)	next	(p)	$p:=$POINTER (p);
(3)	clear	(p_h)	$pT:=p_h-2$; erase (p_h);
(4)	decomp	(p)	**if** DEG $(pA)=-1$ **then goto** ERROR; $p:=$POINTER (pA); POINTER $(pA):=$ POINTER (p); POINTER $(p):=$**nill**;
(5)	set	(p)	**begin pointer** $p1$; head (p); decomp $(p1)$; DEG $(p):=0$; EPOINTER $(p):=$POINTER $(p):=p1$ **end**
(6)	conc	$(p1_h, p2_h)$	**begin pointer** $p3$; $p3:=$EPOINTER $(p1)$; POINTER $(p3):=$POINTER $(p2_h)$; EPOINTER $(p1_h):=$EPOINTER $(p2_h)$; DEG $(p1_h):=$DEG $(p1_h)+$DEG $(p2_h)+1$ **end**
(7)	concs	(p_h, pp_h)	**begin pointer** $p1$; set $(p1)$: conc $(p_h, p1)$; clear $(p1)$; $pp_h:=$EPOINTER (p_h); **end**

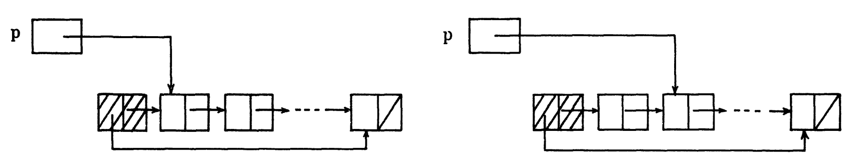
```
(p_h, pp_h)   begin pointer p1;
                 set (p1);   conc (p1, p_h);   HEAD (p_h):
                 =HEAD (p1); clear (p1);
                 pp_h: =POINTER (p_h)
              end

  (p_h)       begin pointer wp, w1p;  wp: =w1p: =p_h;
              M: if POINTER (w1p) =nill
              then goto EXIT else next (w1p);
              if (TAG (w1p) ≠integer) ∧ (TAG (w1p)
              ≠letter) then conc (pA, wp); goto M
       EXIT: end

 (p_h, p)     begin pointer p1; p1: =p_h;
              L: if POINTER (p1) ≠ p then
              begin next (p1); goto L end;
                 POINTER (p1): =POINTER (p);
                 POINTER (p): =nill;
                 DEG (p_h): =DEG (p_h) -1; p1: =p;
                 head (p); POINTER (p): =p1
              end

 (p_h, p)     begin pointer wp, tp, wp1, tp1, sp, spp, gp;
                 head (sp); head (gp); tp1: =p;
                 tp: =wp: =ph;
              L: head (tp1); HEAD(tp1): =HEAD (tp);
              M: if POINTER (wp) =nill then goto R;
                 next (wp); concp (tp1, wp1);.
                 if (TAG (wp) =integer) ∨ (TAG (wp)
                 =letter) then
                   begin NODE (wp1): =NODE (wp);
                   goto M end; concp (sp, spp);
                   INFORMATION (spp): =wp;
                   tp: =INFORMATION (wp); goto L;
              R: if POINTER (sp) ≠nill then
                   begin gp: =POINTER (sp);
```

cut (sp, gp); next(gp);
wp: =INFORMATION (gp);
goto M **end**
end;

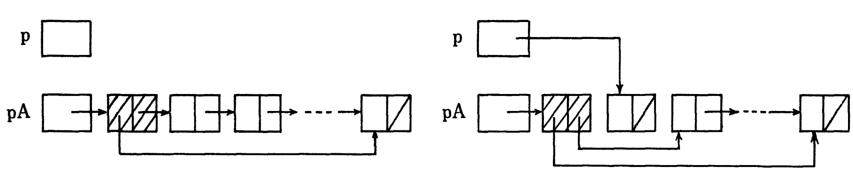
input output

(1) head (p)

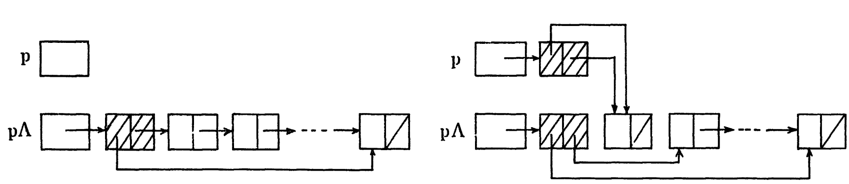
(2) next (p)



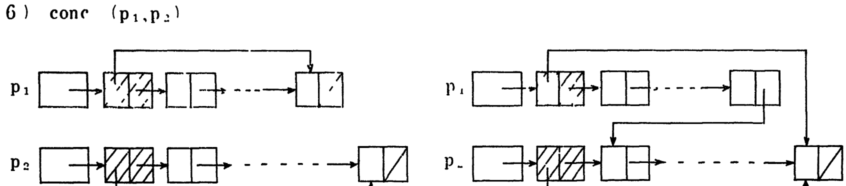
(4) decomp (p)



(5) set (p)



(6) conc (p_1, p_2)



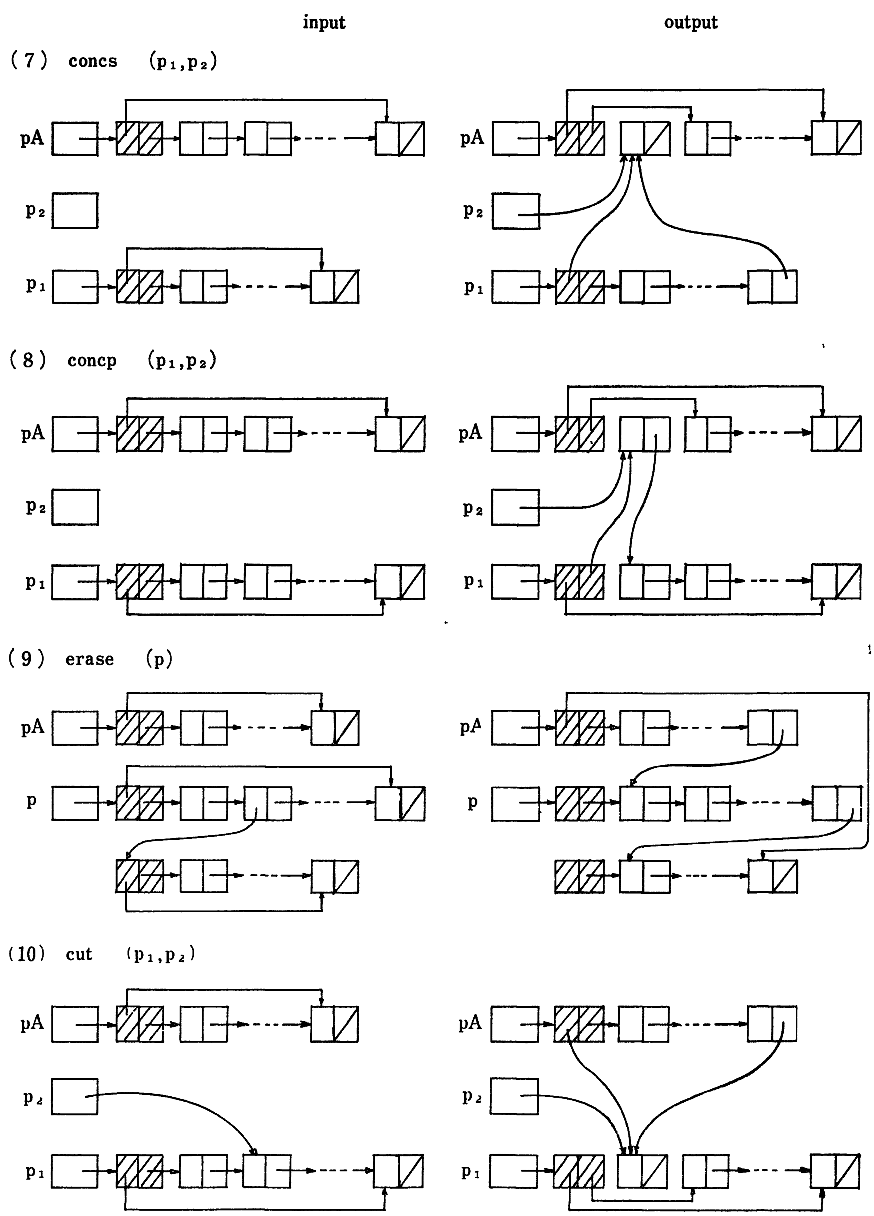


Fig. 8

6. *Lα′*.

1) *Lα* is extended naturally to *Lα′* which have functions of

defining 'structure types' and 'operations between structure types'. A variable declared under a structure type declarer is fixed length node sequence, the type of component nodes is integer, letter, chain, or other data structure. If the type is integer or letter, the INFORMATION part contains integer or letter, however, if it is chain or other data structure, the INFORMATION part contains the head address of the latter, here we notice that for a chain only a head is prepared at the time of its declaration. For example by the structure type declaration

(6.1) **define type pol** $(x)=($**letter** x, **ratn** r, **chain integer** $a)$

(6.2) **define type ratn**$=($**integer** n, **integer** $d)$;,

the list shown in Figure 9 is prepared by $L\alpha'$ compiler, where the head

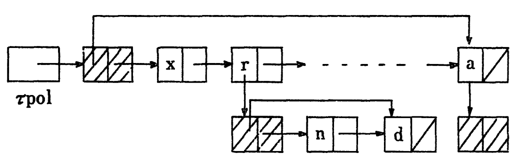


Fig. 9

of this list is refferred by τpol within our system, DEG of the head under a is -1, TYPE of the head of a is **chain integer**, TYPE of the head under r is **ratn**, and TAG of a is **chain.** By the declaration

(6.3) **pol** (x) f, g;,

two lists are copied from the list of Figure 9, and are identified with f and g respectively. At the same time two pointers τf, πf **pol** (x) variable f are generated, and τf, πf contain the head address of f. Therefore $L\alpha'$ system compiles '**pol** $(x)f$;' to

(6.4) **pointer** τf, πf; copy $(\tau pol, \tau f)$; $\pi f:=\tau f$;.

If we use $z:=f.r$ then $L\alpha'$ system generates the following object program of $L\alpha$

(6.5) next (πf); next (πf); $z:=$INFORMATION (πf).

For $f.x$ and $f.a$, 'next (πf); next (πf);' is replaced by 'next (πf)' and 'next (πf); next (πf); next (πf);' respectively. If we use $f.r:=z$ then $z:=$INFORMATION (πf) of (6.5) is replaced by INFORMATION $(\pi f):=z$.

Moreover if we used $w:=f.r.n$ then $L\alpha'$ system generates the following object program of $L\alpha$

$$(6.6)\quad \begin{cases} \text{program of (6.5)} \\ \pi fr:=\tau fr:=\text{INFORMATION}\,(\pi f);\ \text{next}(\pi fr); \\ w:=\text{INFORMATION}\,(\pi fr);. \end{cases}$$

For $f.r.n:=w$ or $w:=f.r.d$ etc, the object program can be obtained similarly.

2) A chain being solely declared or defined within a data structure is compiled to a $L\alpha$ program consisting of a head and two pointers, and we can make any list structures from this head using the subroutines of section 4.

Also we note that a chain is considered conceptually as an array whose lower bound is 0 and upper bound is deg (A), where the latter changes dynamically. Only admissible operation for a chain variable A is an assignment statement

$$A[i]:=E;$$

where E gives only one node which may point another chain, if $i<0$ then the node given by E and auxiliary $|i|-1$ nodes are inserted at the position just after the head of A, and if $i\geq 0$, then the node given by E is inserted just after the i-th node of A and note that $(i-\deg(A))$ auxiliary node are inserted when $i>\deg(A)$. After the insertion, the content of DEG (A) is adjusted accordingly.

In the following we give the translations of the chain declaration and assignment statement to a chain variable and subscribed chain variable in $L\alpha'$ to corresponding $L\alpha$ program.

$L\alpha' \equiv L\alpha$

(6.7) **chain integer** x≡**pointer** τx, πx; head (τx); $\pi x := \tau x$;
TAG1 (τx) := 1; TAG2 (τx) := **integer**;

(6.8) **chain chain integer** y≡**pointer** τy, πy; head (τy); $\pi y := \tau y$;
TAG1(τx) := 2; TAG2(τx) := **integer**;

(6.9) $z[i] := E$≡ASSING (z, E, i); **comment** procedure call of ASSIGN;

(6.10)
```
procedure ASSING (z, E, i);
begin integer k; comment z=x of (1) or y of (2)
if i>DEG (τz) then
begin for k:=DEG (τz)+1 step 1 until i−1 do
    CONCPS (concs, 0); CONCPS (concs, E) end else
if 0≤i∧i≤DEG (τz) then
begin πz:=τz; for k:=0 step 1 until i do
    next (πz); STORE (πz, E) end else
if i<0 then
begin for k:=−1 step −1 until i+1 do
    CONCPS (concp, 0); CONCPS (concp, E) end else
    goto ERROR
end
```

where

(6.11)
```
procedure STORE (x, exp);
comment type of x is chain integer or chain integer, for
  the former exp is integer, for the latter exp is chain
  integer;
if TAG1 (τx)=1 then INFORMATION (πx):=exp else
if TAG1 (τx)=2 then
  begin chain integer s;
     INFORMATION (πx):=τs; s:=exp
  end
```

(6.12)
```
procedure CONCPS (sub, exp); procedure sub;
  begin comment sub=concs or concp;
    sub (τx, πx); TAG (πx):=TAG2 (τx);
```

STORE (x, exp)
integer i; **begin integer** k; **end**;

(6.13) $x[i]$ COMP(x, i) **comment** function designator of COMP;

(6.14) **integer procedure** COMP(x, i); **chain integer** x;
integer i; **begin integer** k;
$\pi x := \tau x$; next(πx);
for $k := 0$ **step** 1 **until** i **do** next (πx);
COMP: =INFORMATION (πx)
end

Similarly we can get the object program of $L\alpha$ for $y[i]$, $y[i][j]$ etc, however these correspondence cannot be given by operation definitions, because these need a representation of infinite number of types, for example COMP of the above must take the form **anytype procedure** COMP(x, i); **chain anytype, integer** i; therefore the object program must be compiled specifically by $L\alpha'$ compiler.

Moreover assignment statements of a data structure to the other data structure both of which have the same structure type is compiled to a program that each part of the right hand side is assigned to the corresponding part of the left hand side, for example '$w := v$:' where w and v are chain integer means w is the copy of v.

3) An operation definition is equivalent to a procedure whose parameters may be data structures or procedures. For example following is the definitions of operator //. (See (3) of Section 8.3)

(6.15) **define operation** $i // j$ **as ratn procedure** $r(i, j)$; **integer** i, j;
begin integer g; $g :=$ **gcm** $((i, j))$; $r.n := i/g$;
$r.d := j/g$ **end**;

We remark that at the end of a block all pointers are checked whether these are declared within this block or not, and all those pointed node sequences or chains are returned to available chain A by subroutine 'erase'.

7. $L\beta$.

1) $L\beta$ is a language for rational functions. First we notice that

the constituents of rational functions are integers, rational numbers, polynomials and rational function themselves, and they are composed into a rational function through formal arithmetic operations. Furthermore, the algorithm, to be programmed, can be described through the processing with respect to these constituents of rational functions and the formal arithmetic operations. Therefore establishing some correspondence between mathematical notations and the editing of lower level subroutined (to manipulate pointers and nodes), we may eliminate all pointers and nodes from the language $L\beta$, so that it becomes easier to read the programs. In fact, all the programs of $L\beta$ are compiled to $L\alpha'$-programs where in the latter programs pointer and node manipulations will appear.

2) For example, the mathematical notation

(7.1) $$f(x)=a_0+a_1x^1+\cdots+a_n\cdot x^n$$

can be understood as follows:

When we know the meaning of operations in the right hand side, namely, the subroutines corresponding to those operations, then the right hand side specifies how to edit these subroutines to make up the subroutine corresponding to the right hand side itself, and the left hand side indicates the proto-type of the calling sequence for the constructed routine, so that $f(3)$, $f(1/2)$ or $f(g(x))$ are the actual calling sequences. Here a problem arises, that is, the meaning of operations in the right hand side may differ according to the type of the value actually substituted to x. Therefore before to enter the constructed routine, it is needed to distinguish the type of the value of x. We also note that $f(x)$ itself may be an actual calling sequence with the actual parameter '**letter** x' where *letter* will be explained later.

Urder these consideration we arrive at the concept 'formula type definition' defined as follows: A typical example of formula type definition and their usage is shown by (7.2).

(7.2) A: **pol** (L) = **letter** $L\times\times$**chain integer** A

where (L) = (**integer**, **ratn**, **pol** (L),

chain integer) **level** 2;

......

B: **pol** (L) f, g;

C: **ratn** s;

D: $s:=f(1/3)$;

......

E: $f(L):=g(L)$.

By $(7.2)A$, we define a declarer **pol** (L), and when f and g are declared as **pol** (L) variables by $(7.2)B$, f and g are identified with a data structure (**letter** L, **chain integer** A) prepared by the right hand side of $(7.2)A$. (See (2) of Section 8.1.) For $(7.2)A$, $L\beta$ system compiles a subroutine $S(F, M)$ which distinguishes whether the type of M is **integer, ratn, pol** (L), or **chain integer** and jumps to those subroutines generated respectively from **integer** $M\times\times$**chain integer** $F.A$, **ratn** M **chain integer** $F.A$, **pol** $(L)M\times\times$**chain integer** $F.A$, or **chain integer** $M\times\times$**chain integer** $F.A$ in operation definitions, where F is a **pol** (L) variable.

f and g must be used with actual parameter of type **integer, ratn, pol** (L), or **chain integer**, the effect of these are a subroutine call of the $S(F, M)$. For example, $f(1/3)$ of $(7.2)D$ is equivalent to $S(f, 1/3)$. However if the type of the actual parameter is **letter** then the data structure generated by $(7.2)B$ is the only result of the call. For example $(7.2)E$ means that $(f.L, f.A):=(g.L, g.A)$.

3) Guided by the same idea explained in 2), we have the concept of operation definition. An operation definition has the form of an equation. The right hand side of this equation specifies how to make up the subroutine corresponding to the operation to be defined from the lower level subroutines which correspond to the operations appearing in the right hand side that are already defined before to reach this operation definition. The other operator symbols in the right hand side which are not defined before to reach this operation definitions are considered merely as separator symbols of the data. The left hand side specifies how to write the operation to be defined, namely, all

the specification of the calling sequence for the subroutine constructed from the right hand side.

More precisely, in the left hand side, operands of the operator to be defined are preceded by formula type name such as **ratn**, and enclosed by brackets. An operator symbol in the left hand side is either in an operand or not in any operand. If an operator symbol is in an operand, then it is considered to be a separator symbol, and the identifiers in the operand are considered as the names of the substructures separated by those separator symbols, and the correspondence is given by the formula type of the operand. If an operator symbol is not in any operand, then this operator symbol represents the operation to be defined.

In the right hand side, if a formula type name appears first, it represents the resulting formula type, therefore this subroutine is function type. If no formula type appears then this subroutine is subroutine type. All identifiers of the right hand side must be given in the left hand side, and this is the correspondence between the identifiers of the calling sequence and the formal parameters of the subroutine to be defined.

4) Operation definitions are typically shown by following three examples.

(1) **integer** $i/\!\!/j=$ **ratn** r; (See (3) of 8.3)
begin integer g; $g:=$ **gcm**$((i, j))$;$/r.n:=i/g$; $r.d:=j/g$ **end**;

$L\beta$ compiler compiles (1) to (1)′, where / is the usual integer division operation, and **gcm** is the unary operator which is defined in (33) of 8.3 for a chain expression (i, j).

(1)′ **define operation** $i/\!\!/j$ **as ratn procedure** $r(i, j)$; **integer** i, j;
begin integer g; $g:=$ **gcm**$((i, j))$; $r.n:=i/g$; $r.d:=j/g$ **end**;

(2) **ratn** $(n1/\!\!/d1)+$**ratn** $(n2/\!\!/d2)=$**ratn** $((n1\times d2+n2\times d2)/\!\!/$
$(d1\times d2)$ (See (2) of 8.3)

$L\beta$ compiler considers $/\!\!/$ of the left hand side as a separator and $n1$, $d1$, $n2$, $d2$ as the renaming of each part of two rational numbers, and considers $/\!/ \times +$ of the right hand side as operators. (2) is compiled to (2)′.

(2)′ **define operation** $r1+r2$ **as ratn procedure** $rad\,(r1, r2)$;
ratn $r1, r2$; **begin integer** $n1, d1, n2, d2, n3, d3$;
$n1{:}=r1.N$; $d1{:}=r1.D$; $n2{:}=r2.N$; $d2{:}=r.2D$;
$n3{:}=n1\times d2+n2\times d1$; $d3{:}=d1\times d2$; $rad{:}=r(n3, d3)$
end; **comment** r is defined at (1)′, rad is a identifier generated by compiler;

(3) $\boldsymbol{D}(\textbf{ratp}\,(L)(r\times(f/g\uparrow n)))$
$=\textbf{ratp}(L)(\times((\boldsymbol{D}(f)\times g+n\times(f\times \boldsymbol{D}(g)))/(g\uparrow(n+1))))$;
(See (21) of 8.3)

The definition of **ratp** (L) is given at (5) of 8.1. Using (5) of 8.1, $L\beta$ compiler can determine the type of $r, f. g, n$, namely r is **ratn**, f, g are **pol** (L), and n is **integer**, therefore $\times/\uparrow(\ )$ of the left hand side are considered as separators of the data structure **ratp** (L). In the right hand side, $\times+\uparrow/(\ )$ $\boldsymbol{D}$ are operators since they are defined in 8.3. (3) is first compiled to (3)′.

(3)′ **define operation** $\boldsymbol{D}\,(h)$ **as ratp** (L) **procedure** $rpdif\,(h)$;
rapt $(L)h$;
begin ratn r; **pol** $(L)f, f1, g$; **integer** $n, n1$;
$r{:}=h.R$; $f{:}=h.F(L)$; $g{:}=h.G(L)$; $n{:}=h.N$;
$f1{:}=D(f)\times g+n\times(f\times D(g))$; $n1{:}=n+1$;
$rpdif{:}=r\times(f1/(g\uparrow n1))$
end

Moreover $\boldsymbol{D}(f)$ and $\boldsymbol{D}(g)$ of (3)′ are replaced by procedure calls for the procedure which are generated by (20) of 8.3. Operations $\times/\uparrow$ of the last statement are not defined in 8.3, therefore these are considered as separators of **ratp** (L).

5) Definitions of transformation rules are typically shown by the following three examples. First we consider the transformation rule:

(1) **integer** $I{:}={:}\textbf{ratn}\,(I/\cdot 1)$;. (See (1) of 8.2)

This rule is compiled to two procedures (1)′ and (1)″, where

(1)′ **ratn procedure** $ri(i)$; **integer** i; **begin** $ri.N:=i$; $ri.D:=1$ **end;**

(1)″ **integer procedure** $ri(r)$; **ratn** r; **if** $r.D\neq 1$ **then** ERROR **else** $ir:=r.N$;.

Procedure (1)′ is called when operations between **ratn** and **integer** occurs or an assignment statement of **integer** to **ratn** occurs, or in the equivalent cases. Procedure (1)″ is called when an assignment statement of **ratn** to **integer**, or the equivalent case occurs. The transformation rules

(2) **integer** $I:=:$ **chain integer** $A:A[0]:=I$; (See (2) of 8.2)

(3) **chain integer** $A:=:$ **pol** $(L)(L\times\times A)$; (See (3) of 8.2)

can be treated similarly and it is left to the readers.

8. The list of all formula types, transformation rules, and operations which are used in Section 9.

The following declarations must be in the block head where the Boolean procedure FROBENIUS of the Section 9 and its call exist.

8.1. List of formula type definitions.

(1) **ratn**=**integer** $N/\cdot D$ **level** 1;

(2) **pol** (L)=**letter** $L\times\times$**chain integor** A
where (L)=(**integer, ratn, pol** (L), **chain integer**)
level 2;

(3) **bp** (L)=**pol** (L) $F\uparrow$**integer** N **level** 3;

(4) **ratf**(L)=**ratn** $R\times$(**pol** (L) F/G) **where** (L)=(**ratn**) **level** 4;

(5) **ratp**(L)=**ratn** $R\times$(**pol** (L) $F/$(**pol** (L) $G\uparrow$**integer** N))
where (L)=(**ratn**) **level** 5;

(6) **polrp** (L, X)=**ratn** $R\times$(**letter** $L\times\times$ **chain ratp** (X) B)
where (L)=(**ratn**) **level** 6;

(7) **bpk**(L)=**bp** $FN(L).\rightarrow$**integer** K;

(8) $\mathbf{qar}(L)=\mathbf{ratn}\,(1/\cdot N)\times\mathbf{pol}\,(L)(Q)++\mathbf{ratn}(1/N)\times\mathbf{pol}(L)(R)$;

(9) $\mathbf{rb}(L)=\mathbf{pol}(L)\ F/G$ **level** 3

8.2. List of transformation rules between different formula types.

(1) **integer** I: $=$: **ratn** $(I/.1)$;

(2) **integer** I: $=$: **chain integer** A: $A[0]:=I$;

(3) **chain integer** A: $=$: **pol** $(L)(L\times\times A)$;

(4) **pol** $(L)F$: $=$: **bp**$(L)(F(L)\uparrow 1)$;

(5) **pol** (L) F: $=$: **ratf** $(L)((1/\cdot 1)\times(F(L)/(L\times\times B)))$: $B[0]:=1$;

(6) **ratn** R: $=$: **ratf** $(L)(R\times((L\times\times A)/(L\times\times B)))$:
$A[0]:=B[0]:=1$;

(7) **ratn** R: $=$: **chain ratn** CR: $CR[0]:=R$;

(8) **pol** $(L)R$: $=$**pol** $(L,X)F$: $F[0](L):=R(L)$;

8.3. List of operation declaration.

In the followings,

for $i:=0$ **step** 1 **until** deg (A) **do** S

is abbreviated as

$i\to A$ *do* S

where S is a statement and A is a chain. Similarly,

for $i:=\deg(A)$ **step** -1 **until** 0 **do** S

is abbreviated as

$i\to A$ *do* S.

(1) **integer** $i\uparrow j=$**ratn** r:
begin integer g; $g:=$**gcm** $((i,j))$; $r.n:=i/g$; $r.d:=j/g$ **end**

(2) $\mathbf{ratn}(n1/\!/d1)+\mathbf{ratn}(n2/\!/d2)=\mathbf{ratn}((n1\times d2+n2\times d1)/\!/(d1\times d2))$

(3) $\mathbf{ratn}(n1/\!/d1)-\mathbf{ratn}(n2/\!/d2)=\mathbf{ratn}((n1\times d2-n2\times d1)/\!/(d1\times d2))$

(4) $\mathbf{ratn}(n1/\!/d1)\times\mathbf{ratn}\,(n2/\!/d2))=\mathbf{ratn}((n1\times n2)/\!/(d1\times d2))$

```
(5)   ratn (n1//d1)/ratn (n2//d2) = ratn ((n1×d2)//(n2×d1))

(6)   chain integer A1+A2=chain integer A3:
      begin integer W1, W2; W1:=A1, W2:=A2;
        if deg (W1)>deg (W2) then A2[deg (W1)]:=0 else
        if deg (W2)>deg (W1) then A1[deg (W2)]:=0;
        i→A1 do A3[i]:= W1[i] + W2[i];
      L: if A3[deg (A3)] =0 ∧ deg (A3)≥1 then
        begin cut (A3, deg (A3)); goto L end
      end comment for cut see (32) of 8.3, cf (10) of Secton 4;

(7)   integer k×chain integer A1=chain integer A2:
      begin integer i; i→A1 do A2[i]:=k×A1[i] end

(8)   chain integer A1×A2=chain integer A3:
      chain integer i; chain integer W; A3[0]:=0;
        i→A1 do begin W:=A1[i]×A2; W[-i]:=0; A3+ W end
      end

(9)   pol(L)(L× ×A1)+pol(L)(L× ×A2)=pol(L)(L× ×(A1+A2))

(10)  pol (L)(L× ×A1)-pol (L)(L× ×A2)
      =pol (L)(L× ×(A1+(-1)×A2))

(11)  pol(L)(L× ×A1)×pol(L)(L× ×A2)=pol(L)(L× ×(A1×A2))

(12)  pol(L)(L× ×A1)% pol(L)(L× ×A2)
      =qar(L)((1/·N)×(L× ×Q)++(1/·N)×(L× ×R))
       comment Q is the quotient, R is the remainder;
      begin integer i, s; chain integer W1, W2;
        s:=deg (A1)-deg (A2); if s<0 then goto ERROR;
        N:=A2(deg (A2))↑(s+1); W1:=inv (A1); W2:=inv (A2);
        for i:=s step -1 until 0 do
          begin integer j; chain integer WR;
            Q[i]:=W1[0]×W2[0]↑s;
            WR:=W2[0]×W1-W1[0]×W2;
            for j:=1 step 1 until deg(WR) do W1[j-1]:=WR[j]
          end;
```

$R := WR$

end

(13) **pol** (L) $F /\!/ G$ $=$ **rb** $(L)(F1/G1)$:

begin pol $(L) H$; **qar** $(L) F11, G11$;

$F11 := F1 \% H$; $G11 := G1 \% H$;

$F1 := F11.Q$; $G1 := G11.Q$

end

(14) **ratf** $(L)((n1/\cdot d1)\times(F1/G1))+$ **ratf** $(L)((n2/\cdot d2)\times(F2/G2))$

$= \mathit{ratf}\,(L)((1/\cdot(d1\times d2))\times((n1\times d2)\times(F1\times G2)$

$+(n2\times d1)\times(F2\times G1))/\!/(G1\times G2)))$

(15) **ratf** $(L)((n1/\cdot d1)\times(F1/G1))-$ **ratf** $(L)((n2/\cdot d2)\times(F2/G2))$

$=$ **ratf** $(L)((1/\cdot(d1\times d2))\times((n1\times d2)\times(F1\times G2)$

$-(n2\times d1)\times(F2\times G1))/\!/(G1\times G2)))$

(16) **ratf** $(L)((n1/\cdot d1)\times(F1/G1))\times$ **ratf** $(L)((n2/\cdot d2)\times(F2/G2))$

$=$ **ratf** $(L)(((n1\times n2)/\!/(d1\times d2))\times((F1\times F2)/\!/(G1\times G2)))$

(17) **ratf** $(L)((n1/\cdot d1)\times(F1/G1))/$ **ratf** $(L)((n2/\cdot d2)\times(F2/G2))$

$=$ **ratf** $(L)(((n1\times d2)/\!/(n2\times d1))\times((F1\times G2)/\!/(F2\times G1)))$

(18) $\boldsymbol{D}($**pol**$(L)(L\times\times A))=$ **pol** $(L)(L\times\times B)$:

comment $\boldsymbol{D}$ is a differential operator;

begin integer i; **for** $i:=1$ **step** 1 **until** $\deg(A)$ **do**

$B[i-1]:=i\times A[i]$ **end**

(19) $\boldsymbol{D}($**ratp**$(L)(r\times(f/(g\uparrow n))))$

$=$ **ratp** $(L)(r\times((D(f)\times g+n\times(f\times D(g)))/(g\uparrow(n+1))))$

(20) $\boldsymbol{D}($**polrp**$(L,X)(R\times(L\times\times B(X))))$

$=$ **polrp** $(L,X)(R\times(L\times\times DB(X)))$:

begin integer i; $i\rightarrow B(X)$**do** $DB(X)[i]:=D(B(X)[i])$ **end**

(21) **ratn** $R\times\times$**chain integer** $A=$ **ratn** $R1$:

begin integer i; $R1:=0$; $i\leftarrow A$ **do** $R1:=A[i]+R1\times R$ **end**

(22) **cfc**(**chain ratn** LR) $=$ **ratn** $(1/\!/G)\times$**chain integer** A;

begin integer i; $G:=1$; $i\rightarrow LR$ **do** $G:=$ **gcm** $((G, LR[i].n))$;

```
    i→LR do A[i]:=G×LR[i]
  end
(23) pol(L)(L× ×B)× ×chain integer A=pol(L)(L× ×C):
  begin integer i; C[0]:=0;
    for i:=deg(A) step −1 until 1 do C:=A[i]×B+A[i−1]
  end
(24) chain bp(L) FAC/Boolean procedure EQRL
  chain chain bp(L) SETDIV:
  begin integer i; chain bp(L) LFAC, LFAC1;
  LFAC:=FAC(L);
    i→LFAC(L) do begin integer j,k,l; k:=l:=0;
      j→LFAC(L) do
        if EQRL(LFAC[i], LFAC[j]) then
        begin SETDIV[i][k]:=LFAC[j]; k:=k+1 end
        else begin LFAC1[l]:=LFAC[j]; l:=l+1 end;
    LFAC:=LFAC1
  end
(25) pol(L)(A)/+(B)=ratn((B[deg(B)−1]−A[deg(A)−1])
  //(deg(B)×B[deg(B)]))
(26) letter L+·integer k=pol(L)(L× ×A): A[0]:=k
(27) chain ratf(L) CF/ratf(L) G=chain ratf(L) RF:
  begin integer i; i→CF do RF[i]:=CF[i]/G end
(28) inv(chain A)=chain B:
  begin integer i; i→B do B[i]:=A[deg(A)−i] end
```

In the following we omit all the procedure bodies, since they are well known.

```
(29) factor (pol(L)P)=chain bp(L)F:
  comment P(L) is factorized as F[0](L)×···×F[deg(F)](L);
(30) gcm(chain integer A)=integer G:
  comment G is a common divisor of (A[0],···,A[deg(A)]);
(31) pgcm(chain pol(L)B):=pol(L)C:
```

comment $C(L)$ is a common divisor of $(B[1](L), \cdots,$
$B[\deg(B)](L))$;

(32) **cut(chain any** A, **integer** $i)$ = **chain any** B:
comment $A[i]$ *is extracted from* $(A[0], \cdots, A[\deg(A)])$, and the remainder is resubscripted and is named B whose length is less 1 length than A.

(33) **sort(chain any** A, **integer** KEY) = **chain any** B:
comment $A[i]$ must contain its substructure KEY, and $(A[0], \cdots,$ $A[\deg(A)])$ is rearranged so that $A[i]$. KEY are not decreasing, the result is $(B[0], \cdots, B[\deg(B)])$, here $\deg(B) = \deg(A)$;

9. Algorithm

Boolean procedure FROBENIUS (R, A); **chain ratf** $(L)R$; **ratn** A;
comment Calculate coefficients of the power series solution of the

$$R[0](X)(X-A)^n y^{(n)} + R[1](X)(X-A)^{n-1} y^{(n-1)} + \cdots + R[n](X)y = 0$$

at a regular singular point A, and if the solution has logarithmic terms then FROBENIUS is true, else FROBENIUS is false. This procedure must be declared in the block which contain all the declarations given in 8.1, 8.2 and 8.3, and must be called within this block;

begin comment This procedure starts at a statement labelled by START;
  chain chain ratf$(L)GM$; **chain pol**$(L)FM$; **polrp**$(L, X)F$;

  chain ratf(L) **procedure** CALCGR(m); **integer** m;
  comment calculate CALCGR $= g_{m-1} f_1(\lambda + m - 1) + \cdots + g_0 f_m(\lambda)$;
  begin integer j;
    $F(L, X) := (1/\cdot m) \times \boldsymbol{D}(F(L, X))$;
    $FM[\deg(FM)+1] := F(L, A)$; CALCGR$[0] := 0/\cdot 1$;
     for $j = 1$ **step** 1 **until** $\deg(FM)$ **do**
    CALCGR: = CALCGR + GM$[m-j] \times$ FM$[j](L + \cdot (m-j))$
  end;

Boolean procedure INTDIF $(A1, B1)$; **pb** $(L)A1$, $B1$;
comment if there exists an integer k such that $A1.F(L) = B1.F(L+k)$

then true;

begin integer ea, ea; **ratn** k; **pol**(L) a, b;

$a(L):=A1.F(L)$; $b(L):=B1.F(L)$; $ea:=\deg(a(L))$;

$eb:=\deg(b(L))$;

if $(ea\neq eb)\vee(a(L)[ea]\neq b(L)[eb])$ **then goto** F; $k:=b(L)/+a(L)$;

if $(k.D=1)\wedge(b(L+\cdot k.N)=a(L))$ **then begin** INTDIF: =**true**;

goto E **end**; F: INTDIF: =**false**

E: **end**;

Boolean procedure CRDPO(PA, PB); **pol**(L) PA, PB;

comment if PA is divisible by PB then true else false;

begin qar$(L)PC$; $PC:=PA\% PB$; CRDPO: =**if** $PC.R=0$ **then true else false end**;

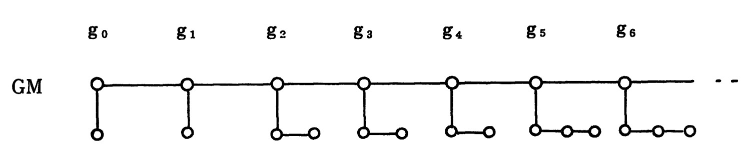
Boolean procedure IFMAKGM(J); **integer** J;

comment if we can construct $g_m(\lambda)$ which satisfies (2.8) for $m=0, 1, 2, \cdots, k_\tau-k_J$, then IFMAKGM=true else false;

begin chain integer PM; **chain ratf**(L) G;

comment $G=g_{m-1}f_1(\lambda+m-1)+\cdots+g_0f_m(\lambda)=h_0(\lambda)+h_1(\lambda)p_1+\cdots+h_\omega(\lambda)p_\omega$, see (2.8) where $h_i(\lambda)$ are rational functions and p_i are free parameter and is represented as $(h_0(\lambda), h_1(\lambda), \cdots, h_\omega(\lambda))$.

$PM=(i_1, \cdots, i_\pi)$ where $\deg(g_{i_l})=\deg(g_{i_{l-1}})+1$.



in the above example $PM=(0, 2, 5, \cdots)$;

procedure CSEPUT(CTR); **integer** CTR;

comment free parameter FPC indicated by the counter CTR of PM is made first and is substituted into all the $g_m(\lambda)$ which contain this parameter;

begin integer i, j, k, l; **chain ratf**(L) FPC, $FPC1$;

$k:=0$; $i\rightarrow G$ **do if** $i\neq CTR$ **then**

```
  begin FPC1[k]:=G[i]; k: k+1 end;
  FPC:=FPC1/((-1)×G[CTR]);
  for i:=CTR step 1 until deg(PM) do
  for j:=PM[i] step 1 until
  if i=deg(PM) then deg(GM) else PM[i+1]-1 do
  begin chain ratf(L) GMC, GMC1;
    l:=0; k→GM[j] do
    if k≠CTR then begin GMC[l]:=GM[j][k]; l:=l+1 end
    else GM[j]:=GMC1+GMC
  end
end;

integer I, PCTR, m1; chain integer KJ;
comment PCTR is a free parameter counter NUK is a global para-
meter of the form
  NUK=((ν1, 0), (ν2, k2), ···, (ντ, kτ)) which was explained in (2.10),
  KJ=(0, k2, ···, kτ). For parameter J, NUK and KJ take the follow-
  ing form NUK=((νJ, kJ),···,(ντ, kτ)), KJ=(k'J, k'J+1, ···, k'τ) as
  (2·10'). g_m is calculated for m:=m1 step 1 until kJ-1. Also
  SPHI is a global parameter;

  m1:=1; PCTR:=0; PM[0]:=0;
  for I:=J+1 step 1 until deg(NUK) do
  KJ[I-(J+1)]:=NUK[I].K-NUK[J].K
    I→KJ do
    begin chain ratf GMFP; integer j, m;
    comment GMFP represents a free parameter of PCTR which has
    the form (0, ···, 0, (1/1)) where the number of 0 is PCTR;
    for m:=m1 step 1 until KJ[I]-1 do
    GM[m]:=CALCGR(m)/FM[0](L+·m);
    comment for/see (27), for +· see (28) of 8.3;
    m1:=KJ[I]+1; G:=CALCGR(KJ[I]);
    if deg(G)=0 then
      if CRDPO(G[0], SPHI) then goto LFP else
      begin IFMAKGM:=false; goto EXIT2 end;
```

for $j:=1$ **step** 1 **until** $deg(G)$ **do**
if $\neg CRDPO(G[j], SPHI)$ **then begin** $CSFUT(j)$; **goto** LFP
end;

LFP: **comment** make free parameter;
$PCTR:=PCTR+1;\ GMFP[PCTR]:=1;$
comment for the latter: $=(1)$ and (7) of 8.2 are used;
$GM(KJ[\mathrm{I}]):=GMFP;\ PM[\deg(PM)+1]:=KJ[I];$
if $I=\deg(KJ)$ **then** $IFMAKGM:=$**true**

$EXIT2$: **end**;

$MAIN$: **integer** k; **pol** $(L)P$; **chain chain bp** (L) $PLGR$;
comment $P(L)=L\cdot(L-1)\cdots(L-k+1)=\lambda(\lambda-1)\cdots(\lambda-k+1)$ of (2.5), $PLGR(L)=((\varphi_1(\lambda)^{\nu_{11}},\cdots,\varphi_1(\lambda-k_{\tau 1})^{\nu_{\tau 1}}),\cdots)$ of (2·10);

$START$: $F(L,X):=0;\ P(L):=1;$
comment for the former (8), and for the latter, (2) and (3) of 8.2 are used;
$k\rightarrow R$ **do begin** $F(L,X):=F(L,X)+R[\deg(R)-k]\times P(L);$
$P(L):=P(L)\times(L+\cdot(-k))$ **end**;
$PLGR:=$**factor**$(F(L,A))/INTDIF;$
comment / is defined by (24) of 8.3;

$k\rightarrow PLGR$ **do**
begin chain bpk $FNK, FNK1$; **chain ratn** NUK; **integer** l, SW;
ratf (L) PHI; **comment** first make $NUK=((\nu 1, 0), (\nu 2, k2),\cdots, (\nu\tau, k\tau))$, $SPHI=\varphi_1(\lambda)$; $l\rightarrow PLGR[k]$ **do** $FNK[l]:=PLGR[k][l]$ $\cdot\rightarrow(PLGR[k][0]/+PLGR[k][l]);$
comment the type of $(PLGR[k][0]/+PLGR[k][l])$ is rational number, but it is transformed to integer, because its denominator $=1$ and the type of $FNK[l]$ is $bpk=pol(L)\uparrow$ integer;
$FNKI:=$**sort**$(FNK, FNK.K);\ PHI(L):=FNK1.[l].FN.F[L];$
$l\rightarrow FNK1$ **do** $NUK[l]:=FNK1[l]N./\cdot(FNK1[l].K-FNK1[0].K);$
$l\rightarrow NUK$ **do**
begin integer lj; **ratf** (L) $SPHI$;
$GM[0]:=1;\ SPHI(L):=PHI(L+\cdot(-NIK[l].K));$

```
    if NUK[l].N=1 then
SIMPROOT: begin
              if l=deg(NUK) then goto NOLOG else
              if IFMAKGM(l) then goto NOLOG else goto LOG
          end else
MULTIROOM: if l=deg(NUK) then goto LOG else
              for lj:=l+1 step 1 until deg(NUK) do
              GM[0]:=GM[0]×NUK.F(L+·(−NUK[lj].K));
LOG: SW:=1;
NOLOG: end;
FROBENIUS:=if SW=1 then true else false end FROBENIUS.
```

10. Results of computations.

The results are listed up in the following pages.

```
PROBREM

( 1)   (X-1)¥2*Y,,+(X-1)*R2(X)*Y,+R1(X)*Y=0

WHERE


                 16   (+    1+    0X+    0X¥2+    0X¥3+    1X¥4)
       R1(X) =-----, ----------------------------------------------
                  1   (+    1+    1X+    1X¥2+    1X¥3)¥2

                  2   (+    0-    5X+    0X¥2+    1X¥3)
       R2(X) =-----, ----------------------------------
                  1   (+    1+    1X+    1X¥2+    1X¥3)

CHARACTERISTIC EQUATION IS

           1L¥2-    3L+    2

       =(    1L-    1)*(    1L+   -2)

                  2   (-    1+    1L)
       GR(L) =-----, ---------------
                  1   (+    1)
                  .
NO LOG TERM APPEARS

PROBREM

( 2)   (X-1)¥2*Y,,+(X-1)*R2(X)*Y,+R1(X)*Y=0

WHERE


                  1   (-    1+    2X+    0X¥2-    2X¥3+    1X¥4)
       R1(X) =-----, ----------------------------------------------
                  1   (+    0+    1X)¥2

              -   1   (+    1+    0X+    1X¥2)
       R2(X) =-----, -------------------------
                  1   (+    0+    1X+    1X¥2)

CHARACTERISTIC EQUATION IS

           1L¥2-    2L+    0

       =(    1L+    0)*(    1L+   -2)

              -   1   (+    0+    1L)
       G1(L) =-----, ---------------
                  2   (-    1+    2L)
```

```
                 1   (+    0+    2L-    7L¥2)
GR(L) =------, ---------------------
                 4   (-    1+    2L)

NO LOG TERM APPEARS

PROBREM

( 3)  (X-2)¥2*Y,,+(X-2)*R2(X)*Y,+R1(X)*Y=0

WHERE

                 1   (-   16+   16X+    0X¥2-    4X¥3+    1X¥4)
R1(X) =------, -----------------------------------------------
                 1   (+    0+    1X)¥2

              -  1   (+    4+    0X+    1X¥2)
R2(X) =------, ----------------------
                 1   (+    0+    2X+    1X¥2)

CHARACTERISTIC EQUATION IS

       1L¥2-    2L+    0

    =(    1L+    0)*(    1L+   -2)

              -  1   (+    0+    1L)
G1(L) =------, ---------------
                 4   (-    1+    2L)

                 1   (+    0+    2L-    7L¥2)
GR(L) =------, ----------------------
                16   (-    1+    2L)

NO LOG TERM APPEARS

PROBREM

( 4)  (X-0)¥2*Y,,+(X-0)*R2(X)*Y,+R1(X)*Y=0

WHERE

                 1   (-    2+    0X+    1X¥2)
R1(X) =------, ----------------------
                 1   (-    1+    0X+    1X¥2)

              -  1   (-    2+    0X+    1X¥2)
R2(X) =------, ----------------------
                 1   (-    1+    0X+    1X¥2)

CHARACTERISTIC EQUATION IS
```

```
      1L¥2-    3L+    2
    =(   1L-    1)*(   1L+  -2)

                  0   (+    0)
    GR(L) =------. ---------
                  1   (+    1)
NO LOG TERM APPEARS
PROBREM
( 5)  (X-0)¥2*Y,,+(X-0)*R2(X)*Y,+R1(X)*Y=0
WHERE

                  1   (-    1+   0X+    1X¥2)
    R1(X) =------. ----------------------
                  1   (+    1)

                  1   (+    1)
    R2(X) =------. ---------
                  1   (+    1)
CHARACTERISTIC EQUATION IS
      1L¥2+    0L-    1
    =(   1L+    1)*(   1L+  -1)

                  0   (+    0)
    G1(L) =------. ---------
                  1   (+    1)

                  1   (+    1)
    GR(L) =------. ---------
                  1   (+    1)
  LOG TERM APPEARS
PROBREM
( 6)  (X-1)¥2*Y,,+(X-1)*R2(X)*Y,+R1(X)*Y=0
WHERE

                  1   (-    1+    1X)
    R1(X) =------. --------------
                  4   (+    0+    1X)
```

```
                 1   (-    3+    4X)
R2(X) =------.  ---------------
                 2   (+    0+    1X)
```

CHARACTERISTIC EQUATION IS

```
      2L¥2-    1L+    0

    =(    1L+    0)*(    2L+   -1)
```

NO LOG TERM APPEARS

PROBREM

```
( 7)  (X-1)¥2*Y,,+(X-1)*R2(X)*Y,+R1(X)*Y=0
```

WHERE

```
              -  12   (+    0+    1X)
R1(X) =------,  ---------------
                  1   (+    1+    1X)¥2

              -   2   (+    0+    1X)
R2(X) =------,  ---------------
                  1   (+    1+    1X)
```

CHARACTERISTIC EQUATION IS

```
      1L¥2-    2L-    3

    =(    1L+    1)*(    1L+   -3)
```

```
                  1   (+    0+    1L)
G1(L) =------,  ---------------
                  2   (-    1+    2L)

              -   1   (+    0+    6L)
G2(L) =------.  ---------------
                  4   (+   24+   12L)

              -   9   (-    7-   16L)
G3(L) =------,  ---------------
                  4   (+1656+1548L)

                 27   (+1614- 259L+1676L¥2+ 487L¥3-   32L¥4)
GR(L) =------,  ----------------------------------------
                  2   (-9744+2464L+5472L¥2+7152L¥3)
```

LOG TERM APPEARS

PROBREM

```
( 8)  (X-2)¥2*Y,,+(X-2)*R2(X)*Y,+R1(X)*Y=0

WHERE


                 -    2  (+    1)
       R1(X) =------, ---------------
                      1  (-    1+    1X)¥2


                      1  (+    0+    1X)
       R2(X) =------, ---------------
                      1  (-    1+    1X)

CHARACTERISTIC EQUATION IS

          1L¥2+    1L-    2

       =(    1L-    1)*(    1L+    2)


                 -    1  (+    4-    1L)
       G1(L) =------, ---------------
                      1  (+    2+    2L)


                 -    1  (-   22-    4L)
       G2(L) =------, ---------------
                      1  (+   28+   12L)


                      4  (+ 274+ 158L-    11L¥2-    5L¥3)
       GR(L) =------, -------------------------------
                      1  (+   56+   80L+   24L¥2)

   LOG TERM APPEARS

PROBREM

( 9)  (X+3)¥2*Y,,+(X+3)*R2(X)*Y,+R1(X)*Y=0

WHERE


                      1  (+    3+    1X)
       R1(X) =------, ---------------
                      1  (+    4+    1X)


                      1  (+    6+    1X)
       R2(X) =------, ---------------
                      1  (+    2+    1X)

CHARACTERISTIC EQUATION IS

         -1L¥2+    4L+    0
```

```
      =(    1L+    0)*(   -1L+    4)
 NO LOG TERM APPEARS
 PROBREM
 (10)  (X-4)¥2*Y,,+(X-4)*R2(X)*Y,+R1(X)*Y=0
 WHERE

              -    2   (+    0+    1X)
      R1(X) =------, ---------------
                   1   (-    2+    1X)

                   2   (-    3+    1X)
      R2(X) =------, ---------------
                   1   (-    2+    1X)
 CHARACTERISTIC EQUATION IS
         1L¥2+    0L-    4
      =(    1L+    2)*(    1L+   -2)

              -    1   (+    2+    1L)
      G1(L) =------, ---------------
                   2   (+    1+    2L)

              -    1   (-   20-   10L)
      G2(L) =------, ---------------
                   4   (+   36+   12L)

              -    1   (+  900+  450L)
      G3(L) =------, ---------------
                   4   (+3204+1548L)

                   1   (-7884-2652L-9241L¥2-0051L¥3-3536L¥4)
      GR(L) =------, ------------------------------------------
                   2   (+5344+4864L+6928L¥2+7152L¥3)
   LOG TERM APPEARS
 JOB COMPLETED
1513LINE   8MIN.  24SEC. THIS JOB.
```

References

[1] Hukuhara, M., Solutions of linear ordinary differential equations II, Iwanami, 1942 (Japanese).
[2] van der Waerden, Algebra I, II, 1937.
[3] McCarthy, LISP 1.5, Programmers Manual.
[4] Naul, P. (Ed.), Report on the algorithmic language ALGOL 60, Comm. ACM **3** (1960), 299-314.
[5] Sammet, J. E., Survey of Formula Manipulation, Comm. ACM, **9** (1966), 555-569.
[6] Brown, E. S. et al., The ALPAK System for Nonnumerical Algebra I, II, III, the Bell system technical Journal **1** (1963); II, III (1964).
[7] Galler, B. A., and J. A. Perlis, A proposal for definitions in ALGOL. Comm. ACM **10** (1967), 204-219.
[8] Collins G. E., PM, A system for Polynomial manipulation comm. ACM **9** (1966), 578-589.
[9] Joel, M., Solution of systems of polynomial equations by elimination, Comm. ACM **9** (1966), 634-637.