STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

AFDELING MATHEMATISCHE STATISTIEK

S 413

AN INPUT SYSTEM FOR LINEAR PROGRAMMING PROBLEMS

part 2

implementation

by
Jac.M. Anthonisse

preliminary 5

August 1969

Summary

An Algol 60 computer program is presented that accepts as input the mathematical formulation of a linear programming problem and generates as output the standardized matrix of constraints.

Introduction

Any linear programming problem can be written as

$$\text{maximize} \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1, \ldots, m_1)$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (i = m_1+1, \ldots, m)$$

$$x_j \geq 0 \qquad (j = 1, \ldots, n).$$

The formulation of a practical problem as a LP problem often leads to more complex formulas. However, by re-indexing the variables and constraints the above form can be found.

Computer-programs that solve LP problems require as input the coefficients c_j, b_i, a_{ij} of the problem. Usually these coefficients must be given column after column, i.e. input consists of the numerical values of

$$\begin{array}{lllll}
n & m_1 & m & & \\
c_1 & a_{11} & a_{21} & \cdots & a_{m1} \\
\cdot & & & & \\
\cdot & & & & \\
\cdot & & & & \\
c_j & a_{1j} & a_{2j} \cdots & a_{ij} \cdots & a_{mj} \\
\cdot & & & & \\
\cdot & & & & \\
\cdot & & & & \\
c_n & a_{1n} & a_{2n} & \cdots & a_{mn} \\
 & b_1 & b_2 \cdots & b_i \cdots & b_m .
\end{array}$$

Many programs accept a condensed form of this scheme, instead of

$$c_j \qquad a_{1j} \qquad a_{2j} \cdots a_{ij} \cdots a_{mj}$$

only the non-zero coefficients must be given, each preceded by a row index.

Whatever scheme of input is used, the step from the (compact) mathematical formulation of a problem to that particular scheme cannot be avoided. This step consists of mainly simple but tedeous and error-prone calculations.

Such tasks are best left to a computer.

General Outline

The Algol 60 computer-program presented in this report accepts as input the mathematical formulation of a (mixed) LP problem as defined in [1], with the exclusion of post-optimization and parametrization. Each variable is assumed to have lower bound zero, unless specified otherwise. The input text is reproduced by a printer. Indices 1, 2, ..., M are assigned to constraints and indices 1, 2, ..., N are assigned to the variables of the LP problem in accordance with the following standardized form of a (mixed) LP problem:

$$\text{maximize} \quad \sum_{j=1}^{N} c_j x_j$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{j=1}^{N} a_{ij} x_j \leq b_i && (i = 1, \ldots, M_1) \\
& \sum_{j=1}^{N} a_{ij} x_j = b_i && (i = M_1+1, \ldots, M) \\
& l_j \leq x_j \leq u_j && (j = 1, \ldots, N) \\
& x_j = \text{integer} && (j = 1, \ldots, N_1).
\end{aligned}$$

A list, giving the correspondence between the indices and the constraints as identified in the mathematical formulation, is printed. A similar list is printed for the variables.

A tape is punched, containing the numerical values of

$$N_1 \qquad N \qquad M_1 \qquad M \,,$$

followed by

$$b_1 \qquad b_2 \quad \ldots \quad b_M$$

and, for each variable,

$$j \quad l_j \quad u_j \quad r_1 \quad a_{r_1 j} \quad \ldots \quad r_p \quad a_{r_p j} \quad M{+}1 \quad c_j \,,$$

where

j = index of the variable,

and $a_{ij} = 0$ if $i \notin \{r_1, \ldots, r_p\}$.

This tape can serve as input for a computer-program that solves the (mixed) LP problem.

The Algol 60 program does not contain an $M \times N$ array with the values a_{ij}, the coefficients of a single variable are computed and punched, then this process is repeated for the next variable.

Input and Output

As mentioned above, the input for the Algol 60 program consists of the mathematical formulation of a (mixed) LP problem as defined in [1]. This formulation of the problem must be typed on a Flexowriter, which simultaneously punches a tape that serves as input for the Electrologica X8 computer. The Algol 60 program reads this tape by means of the procedure RESYM. Table I gives the correspondence between:
(α) the value of RESYM and
(β) the Flexowriter symbol.

(α)	(β)	(γ)
0	0	0
•	•	•
•	•	•
•	•	•
9	9	9
10	a	A
•	•	•
•	•	•
•	•	•
35	z	Z
37	A	$\underset{\sim}{A}$
•	•	•
•	•	•
•	•	•
62	Z	$\underset{\sim}{Z}$
64	+	+
65	−	−
66	×	×
67	/	/
70	=	=
72	<	<
74	>	>
76	¬	¬
79	∨	∨
80	∧	∧

87	,	,
88	.	.
89	$_{10}$	$_{10}$
90	:	:
91	;	;
93	<space>	<space>
94	<undefined>	o
95	<undefined>	\
96	<undefined>	#
98	(	(
99	)	)
100	[	[
101	]	]
118	<tabulate>	<tabulate>
119	<new line carriage return>	<new line carriage return>
120	'	'
121	"	"
122	?	?
126	_	_
127	|	|

table I

The contents of the input tape are printed by means of the procedure PRSYM(v). Table I gives the correspondence between
(α) the value of v and
(γ) the symbol that is printed.
Furthermore the Algol 60 program contains calls of the following output procedures:

RUNOUT : punching a piece of blank tape,

FIXP(n,m,i): punching the numerical value of i in fixed point notation, with at most n digits before and m digits after the decimal point,

PUNLCR	: punching a <new line carriage return>,
PUNCH(r)	: punching the numerical value of r in floating point notation,
NEWPAGE	: shifting the paper to the first line of a new page,
print(r)	: printing the numerical value of r in floating point notation,
PRINTTEXT(s)	: printing the string s.

Some symbols occurring in the formulation of the LP problem consist of two or more punchings, these are listed under (β) in table II.

(α)	(β)
69	$\uparrow$
73	$\leq$
75	$\geq$
81	$\underline{\text{index}}$
85	MAXIMIZE
86	MINIMIZE
102	$\nless$
103	$\ngtr$
106	INIT
107	$\underline{\text{real}}$
108	$\underline{\text{integer}}$
109	POST
111	RANGE
113	PARA
114	OPEN
115	CLOSE
128	$\underline{\text{formula}}$
129	$\underline{\text{discrete}}$
130	$\underline{\text{continuous}}$
131	SUM

table II

Punching of a _ (underlining) or a | (bar) does not cause a displacement of the carriage of the Flexowriter. Thus the symbol ↑ consists of a | followed by ∧ and ≤ consists of _ followed by <.

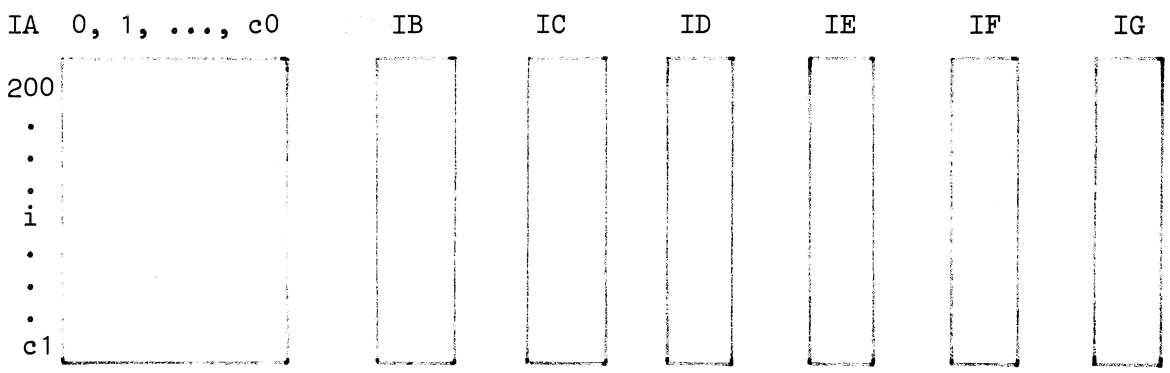
Storing the LP-problem

In the outermost block of the Algol 60 program the integers c0, c1, ..., c7 are declared, and values are assigned to these integers. The integers are used in the declarations of arrays in the second block of the program, and have the following interpretations:

c0 = upperbound for the length of identifiers in the LP problem,
c1 = 199 + upperbound for the number of identifiers in the LP problem,
c2 = 1 + upperbound for the number of constraint descriptions in the LP problem,
c3 = upperbound for the length of the LP problem,
c4 = c1 + 1 + upperbound for the number of numerical values stored as reals,
c5 = 1 + upperbound for the total length of all strings in the LP problem,
c6 = c4 + 1 + upperbound for the number of formula elements,
c7 = c1 + 1 + upperbound for the number of integers to be stored.

The structural part of the LP problem is stored in the array P. However, most numerical values and real formulas are stored elsewhere and then P contains only a pointer to the appropriate value or formula.

IA, IB, ..., IG contain information about the identifiers of the LP problem:

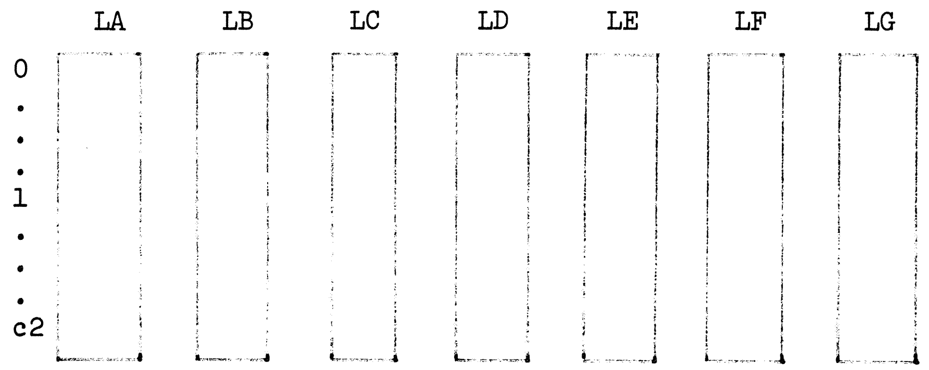


Each identifier corresponds to a unique row of the above scheme.

IA[i,0]	= l = length of the identifier (length $\leq$ c0),
IA[i,1],...,IA[i,l]	= letters and digits constituting the identifier,
IB[i]	= type of the identifier,
	= 1, 2, 3, 4, 5, or 6 if the identifier is of type index, integer, real, formula, discrete variable or continuous variable respectively,
IC[i]	= number of subscripts of the identifier,
ID[i]	= pointer to the domain of a subscripted,
IE[i]	= pointer to the numerical value(s),
IF[i]	= number of numerical values corresponding with the identifier,
IG[i]	= auxiliary storage.

Instead of the letters and digits constituting the identifier the value of i is used throughout the Algol 60 program.

LA, LB, ..., LG contain information about the objective function and constraint descriptions of the LP problem.



The first row of this scheme (l = 0) corresponds to the objective function, each constraint description corresponds with a unique row (1 $\leq$ l $\leq$ c2).

LA[l] = pointer to the constraint description or objective function,

LB[l] = type of the constraint description ($\leq$, $\geq$ or =),

LC[l] = pointer to the right hand side of the constraint description,

LD[l] = pointer to the constraint identification,

LE[l] = pointer to the domain of the constraint description,

LF[l] = number of simple domains constituting the domain,

LG[l] = number of constraints given by the contraints description.

Numerical values of type integer are stored in II. All strings are stored in an array string. Numerical values of type real and numerical values that are explicitly given in the structural part are stored in RR.

FA, FB, FC contain real formulas.

The integers str, lf, p, id, con, num, f are pointers to the last elements of string, LA, ..., LG, P, IA, ..., IG, II, RR, FA, ..., FC that have been used.

Algol 60 program

The tape containing the Algol 60 program was printed by a Flexowriter, the resulting text is reproduced here.
A number of problems was successfully processed by the program, however, it still may contain errors.
The efficiency of the program can certainly be improved.

A future report will contain an extended syntactical definition of LP problems and a corresponding extended and improved implementation. That report will also give a more detailed description of the Algol 60 program.

```
begin comment Input System Linear Programming Problems, jma 110769;
Boolean test;
integer c0, c1, c2, c3, c4, c5, c6, c7;
test:= true; c0:= 15; c1:= 499; c2:= 50; c3:= 1000; c4:= 999;
c5:= 500; c6:= 2000; c7:= 2500;
begin integer array IA[200:c1,0:c0], IB, IC, ID, IE, IF, IG[200:c1],
      LA, LB, LC, LD, LE, LF, LG[0:c2], P[1:c3], II[c1 + 1:c7],
      string[1:c5], FA, FB, FC[c4 + 1:c6];
      real array RR[c1 + 1:c4];
      integer lettera, letterd, lettern, letterr, letters, lettert,
      letterA, letterC, letterI, letterM, letterO, letterP, letterR,
      letterS, sigma, letterU, plus, min, mult, div, power, equal,
      smaller, less, greater, more, hat, index, MAXIMIZE, MINIMIZE,
      comma, point, lowten, colon, semicolon, space, par, rap, sub,
      bus, openstring, closestring, INIT, real, integer, POST,
      RANGE, PARA, OPEN, CLOSE, tab, nlcr, line, bar, formula,
      discrete, continuous, sum, str, lf, p, id, con, num, f, mem,
      cursym, con1, num1, str1, objective, m, n, N1, N, M1, M, dmax;
      Boolean skipcap, skipline, error, model;

      procedure start;
      begin integer i, j;
       for i:= 200 step 1 until c1 do
       begin for j:= 1 step 1 until c0 do IA[i,j]:= 93;
            IB[i]:= IC[i]:= ID[i]:= IE[i]:= IF[i]:= IG[i]:= 0;
       end;
       for i:= 0 step 1 until c2 do LA[i]:= LB[i]:= LC[i]:=
       LD[i]:= LE[i]:= LF[i]:= LG[i]:= 0;
       for i:= 1 step 1 until c3 do P[i]:= 0;
       for i:= c1 + 1 step 1 until c7 do II[i]:= 0;
       for i:= 1 step 1 until c5 do string[i]:= 93;
       for i:= c4 + 1 step 1 until c6 do FA[i]:= FB[i]:= FC[i]:= 0;
       for i:= c1 + 1 step 1 until c4 do RR[i]:= 0; lettera:= 10;
       letterd:= 13; lettern:= 23; letterr:= 27; letters:= 28;
       lettert:= 29; letterA:= 37; letterC:= 39; letterI:= 45;
       letterM:= 49; letterO:= 51; letterP:= 52; letterR:= 54;
       letterS:= sigma:= 55; letterU:= 57; plus:= 64; min:= 65;
       mult:= 66; div:= 67; power:= 69; equal:= 70; smaller:= 72;
       less:= 73; greater:= 74; more:= 75; hat:= 80; index:= 81;
       MAXIMIZE:= 85; MINIMIZE:= 86; comma:= 87; point:= 88;
       lowten:= 89; colon:= 90; semicolon:= 91; space:= 93;
       par:= 98; rap:= 99; sub:= 100; bus:= 101; openstring:= 102;
       closestring:= 103; INIT:= 106; real:= 107; integer:= 108;
       POST:= 109; RANGE:= 111; PARA:= 113; OPEN:= 114;
       CLOSE:= 115; tab:= 118; nlcr:= 119; line:= 126; bar:= 127;
       formula:= 128; discrete:= 129; continuous:= 130; sum:= 131;
       str:= lf:= p:= 0; id:= 199; con:= num:= c1; f:= c4;
       mem:= - 1; model:= error:= skipcap:= skipline:= false;
      end;
```

```
procedure ER(b, s); value b; Boolean b; string s; if b then
begin error:= true; NLCR; PRINTTEXT(s); goto exit end;


Boolean procedure letter; letter:= cursym ≥ 10 ∧ cursym ≤ 35;


Boolean procedure digit; digit:= cursym ≥ 0 ∧ cursym ≤ 9;


real procedure unsignednumber;
begin integer s, w;
 real p, v;
 s:= w:= 0; v:= 0; p:= 1;
L1:  if cursym = point then goto dec;
 if cursym = lowten then goto exp;
 if ¬digit then goto exit; v:= v × p + cursym; p:= 10;
 cursym:= nextsym; goto L1;
dec: p:= 10;
L2:  cursym:= nextsym; if cursym = lowten then goto exp;
 if ¬digit then goto exit; v:= v + cursym / p; p:= p × 10;
 goto L2;
exp: if p = 1 then v:= 1; w:= 0; p:= 1; s:= 1;
 cursym:= nextsym; if cursym = min then s:= - 1;
 if digit then goto L4;
L3:  cursym:= nextsym; if ¬digit then goto exit;
L4:  w:= w × p + cursym; p:= 10; goto L3;
exit: unsignednumber:= v × 10 ↑ (s × w)
end;


integer procedure nextsym;
begin integer c, k, s;

 procedure out(f); integer f;
 begin s:= f; goto exit end;


 Boolean procedure capital; capital:= s ≥ 37 ∧ s ≤ 62;

read: if mem ≠ - 1 then
     begin s:= mem; mem:= - 1; go to tst end;
 s:= RESYM; PRSYM(s);
tst: if s = space ∨ s = nlcr ∨ s = tab then goto read;
 if capital then goto cap; skipcap:= false;
 if s = line then goto lined; skipline:= false;
 if s = bar then goto barred; goto exit;
barred: s:= RESYM; PRSYM(s); ER(s ≠ hat ∧ s ≠ smaller,
 ⍃ |not followed by ∧ or < ⍄); if s = hat then out(power);
 if s = smaller then out(openstring);
lined: s:= RESYM; PRSYM(s); if skipline then goto read;
 if s = smaller then out(less);
 if s = greater then out(more);
 for k:= 1, 2, 3, 4 do
 begin s:= RESYM; PRSYM(s) end;
 skipline:= true; if s = letterd then out(index);
 if s = lettern then out(continuous);
 if s = lettera then out(real);
 if s = lettert then out(integer);
 if s = letterr then out(formula);
 if s = letters then out(discrete); ER(true,
 ⍃  followed by wrong symbols⍄); goto read;
cap: if skipcap then goto read; if s = letterS then
 begin s:= RESYM; PRSYM(s); if s = letterU then
      begin skipcap:= true; out(sum) end
      else
      begin mem:= s; out(sigma) end
 end;
 skipcap:= true; if s = letterC then out(CLOSE);
 if s = letterI then out(INIT);
 if s = letterO then out(OPEN);
 if s = letterR then out(RANGE); if s = letterM then
 begin skipcap:= false; s:= RESYM; PRSYM(s); skipcap:= true;
      if s = letterA then out(MAXIMIZE);
      if s = letterI then out(MINIMIZE)
 end;
 if s = letterP then
 begin skipcap:= false; s:= RESYM; PRSYM(s); skipcap:= true;
      if s = letterA then out(PARA);
      if s = letterO then out(POST)
 end;
 ER(true, ⍃wrong captal letters⍄); goto read;
exit: nextsym:= s
end nextsym;

procedure readstring;
begin
read: cursym:= RESYM; PRSYM(cursym); if cursym = bar then
 begin cursym:= RESYM; PRSYM(cursym);
      if cursym = greater then goto exit; str:= str + 1;
      string[str]:= bar
 end;
 str:= str + 1; string[str]:= cursym; goto read;
exit:
end;


procedure structuralpart;
```

```
begin integer i, v;
 ER(nextsym ≠ OPEN, ‹ OPEN missing ›);
 ER(nextsym ≠ openstring, ‹identification missing›);
 readstring; cursym:= nextsym; declarationpart;
 model:= true; objectivepart; constraintspart; con1:= con;
 num1:= num; str1:= str; LD[lf + 1]:= str;
 for i:= LA[0] + 1 step 1 until p do
 begin v:= P[i]; if v < 200 ∨ v > c1 then go to nexti;
      if IB[v] = 4 then P[i]:= IE[v];
 nexti:
 end;
end;


procedure declarationpart;
begin integer type, i, j;
again: type:= if cursym = index then 1 else if cursym =
 integer then 2 else if cursym = real then 3 else if cursym
 = formula then 4 else if cursym = discrete then 5 else if
 cursym = continuous then 6 else 0; if type ≠ 0 then
 begin list(type); cursym:= nextsym; goto again end;
 for i:= 200 step 1 until id do if IB[i] = 0 then
 begin type:= 1; NLCR;
      for j:= 1 step 1 until c0 do PRSYM(IA[i,j])
 end;
 ER(type ≠ 0, ‹declarations missing›);
exit:
end;


procedure list(type); value type; integer type;
begin integer t, i, j;
new: cursym:= nextsym; ER( ¬letter, ‹wrong symbol›);
 p:= p + 1; i:= P[p]:= identifier(false); t:= IB[i];
 ER(t ≠ 0 ∧ t ≠ type, ‹wrong type›); IB[i]:= type;
tst: if cursym = semicolon then goto exit;
 if cursym = comma then goto new; ER(type = 1,
 ‹wrong symbol›);
 ER(cursym ≠ sub ∧ (cursym ≠ equal ∨ type ≠ 4),
 ‹wrong symbol›); t:= p; if cursym = equal then
 begin cursym:= nextsym; IE[i]:= realformula(t);
      ER(t ≠ 0 ∨ cursym = sigma, ‹wrong realformula›);
      for i:= 200 step 1 until id do if IB[i] = 1 then
      IF[i]:= 0; goto tst
 end;
nxt: cursym:= nextsym; ER( ¬letter, ‹wrong symbol›);
 p:= p + 1; j:= P[p]:= identifier(true);
 ER(if j ≠ 0 then IB[j] ≠ 1 else true,
 ‹wrong identifier›); IF[j]:= 2;
 if cursym = comma then goto nxt; ER(cursym ≠ bus,
 ⊀] missing⊁); t:= p - t;
 if IC[i] = 0 then IC[i]:= t else ER(IC[i] ≠ t,
 ⊀wrong number of subscripts⊁); ID[i]:= p; cursym:= nextsym;
 ER(IC[i] ≠ domain, ⊀wrong domain⊁); cursym:= nextsym;
 goto tst;
exit:
end;


integer procedure identifier(known); Boolean known;
begin integer i, t, s;
 integer array H[1:c0];
 t:= 0;
tp1: if  ¬(letter ∨ digit) then goto tst; t:= t + 1;
 if t ≤ c0 then H[t]:= cursym; cursym:= nextsym; goto tp1;
tst: if t > c0 then t:= c0;
 for i:= 200 step 1 until id do
 begin if IA[i,0] ≠ t then goto nexti;
      for s:= 1 step 1 until t do if IA[i,s] ≠ H[s] then
      goto nexti; goto exit;
 nexti:
 end;
 i:= 0; ER(known, ⊀identifier unknown⊁); i:= id:= id + 1;
 IA[i,0]:= t;
 for s:= 1 step 1 until t do IA[i,s]:= H[s];
exit: identifier:= i
end;


procedure constraintspart;
begin integer v;
again: for v:= 200 step 1 until id do if IB[v] = 1 then
 begin ER(IF[v] = 2, ⊀domain missing ⊁); IF[v]:= 0 end;
 if cursym ≠ openstring then goto exit; lf:= lf + 1;
 LD[lf]:= str; readstring; cursym:= nextsym; LA[lf]:= p;
 linearform(false);
 ER(cursym ≠ equal ∧ cursym ≠ less ∧ cursym ≠ more,
 ⊀type of constraint missing⊁); LB[lf]:= cursym;
 cursym:= nextsym; LC[lf]:= p; p:= p + 1;
 P[p]:= realformula(v); ER(v ≠ 0 ∨ cursym = sigma,
 ⊀variable or sigma in right hand side⊁);
 if cursym ≠ par then goto again; LE[lf]:= p; v:= domain;
 ER(v < 0, ⊀wrong domain⊁); LF[lf]:= v; cursym:= nextsym;
 goto again;
exit:
end;


procedure initialization;
begin integer l;
```

```
 cursym:= nextsym; ER(cursym ≠ openstring,
 ⊀identification missing⊁); str:= str1; readstring;
 cursym:= nextsym; con:= con1; num:= num1; portions;
exit:
end;


procedure linearform(simple); Boolean simple;
begin integer i, d;
new: if cursym = openstring ∨ cursym = less ∨ cursym =
 more ∨ cursym = equal then goto exit; p:= p + 1;
 P[p]:= if cursym = min then min else plus;
 if cursym = plus ∨ cursym = min then cursym:= nextsym;
 p:= p + 1; P[p]:= realterm(i);
 ER(simple ∧ cursym = sigma, ⊀sigma not allowed⊁);
 if cursym ≠ sigma then goto L2; p:= p + 1; P[p]:= sigma;
 p:= p + 1; P[p]:= SIGMA; p:= p + 1; P[p]:= rap; goto new;
L2:  ER(i = 0, ⊀variable or sigma missing⊁);
 ER(IB[i] ≠ 5 ∧ IB[i] ≠ 6, ⊀wrong identifier⊁); p:= p + 1;
 P[p]:= i; if IC[i] = 0 then goto new; ER(cursym ≠ sub,
 ⊀[ missing⊁); d:= subscripts; ER(d ≠ IC[i],
 ⊀wrong number
       of subscripts⊁); ER(cursym ≠ bus,
 ⊀] missing⊁); cursym:= nextsym;
 if cursym = plus ∨ cursym = min then goto new;
exit:
end;


integer procedure subscripts;
begin integer j, d;
 d:= 0;
again: d:= d + 1; cursym:= nextsym; p:= p + 1;
 P[p]:= realformula(j); ER(j ≠ 0,
 ⊀variable in subscript⊁);
 if cursym = comma then goto again; ER(cursym ≠ bus,
 ⊀wrong symbol⊁);
exit: subscripts:= d
end;


integer procedure SIGMA;
begin integer k, i, j;
 i:= 1; ER(cursym ≠ sigma, ⊀sigma missing⊁);
 cursym:= nextsym; ER(cursym ≠ par, ⊀(      missing⊁);
 cursym:= nextsym; ER( ¬letter, ⊀ wrong symbol⊁); p:= p + 1;
 j:= P[p + 1]:= identifier(true); ER(j = 0,
 ⊀unknown identifier⊁); ER(IB[j] ≠ 1, ⊀wrong identifier⊁);
 ER(IF[j] ≥ 1, ⊀ index not allowed⊁); k:= j; IF[j]:= 1;
 ER(cursym ≠ comma, ⊀,     missing⊁); cursym:= nextsym;
 P[p]:= realformula(j); ER(j ≠ 0, ⊀variable in bound⊁);
 ER(cursym ≠ comma, ‹,     missing›); cursym:= nextsym;
 p:= p + 2; P[p]:= realformula(j); ER(j ≠ 0,
 ‹ variable in bound›); ER(cursym ≠ comma,
 ‹,     missing›); cursym:= nextsym;
 if cursym = sigma then i:= i + SIGMA else linearform(true);
 ER(cursym ≠ rap, ‹)
       missing›); cursym:= nextsym;
 IF[k]:= − 1;
exit: SIGMA:= i
end;


integer procedure domain;
begin integer d, j;
 d:= 0; ER(cursym ≠ par, ‹(  missing›);
again: p:= p + 1; cursym:= nextsym; d:= d − 1;
 P[p]:= realformula(j); ER(j ≠ 0, ‹variable in bound›);
 ER(cursym ≠ less, ‹ < missing›); cursym:= nextsym;
 p:= p + 1; j:= P[p]:= identifier(true);
 ER(IB[j] ≠ 1 ∨ IF[j] = − 1, ‹wrong identifier›); IF[j]:= 1;
 ER(cursym ≠ less, ‹ < missing›); cursym:= nextsym;
 p:= p + 1; P[p]:= realformula(j); ER(j ≠ 0,
 ‹variable in bound›); if cursym = comma then goto again;
 ER(cursym ≠ rap, ‹) missing›);
 for j:= 200 step 1 until id do if IB[j] = 1 then
 begin ER(IF[j] = 2, ‹bounds missing›); IF[j]:= 0 end;
 d:= − d;
exit: domain:= d
end;


procedure objectivepart;
begin ER(cursym ≠ MAXIMIZE ∧ cursym ≠ MINIMIZE,
 ‹objective missing›);
 objective:= if cursym = MAXIMIZE then 1 else − 1;
 LA[0]:= p; LB[0]:= equal; cursym:= nextsym;
 cursym:= nextsym; linearform(false); LC[0]:= p;
exit:
end;


integer procedure realformula(ident); integer ident;
begin integer w, v;
 w:= v:= f:= f + 1;
 FB[v]:= if cursym = min then min else plus;
 if cursym = plus ∨ cursym = min then cursym:= nextsym;
again: FC[v]:= v:= f:= f + 1; FA[v]:= realterm(ident);
 ER(ident ≠ 0 ∨ cursym = sigma,
 ‹variable or sigma in formula›);
 if cursym ≠ plus ∧ cursym ≠ min then goto exit;
 FB[v]:= cursym; cursym:= nextsym; goto again;
```

```
exit: if v = w + 1 ∧ FB[w] = plus ∧ FA[v] ≤ c4 then
 begin f:= f - 2; realformula:= FA[v];
      FA[v]:= FB[v - 1]:= FC[v - 1]:= 0
 end
 else realformula:= w
end;


integer procedure realterm(ident); integer ident;
begin integer w, v;
 w:= v:= f:= f + 1;
again: FA[v]:= realfactor(ident);
 if (cursym ≠ mult ∧ cursym ≠ div) ∨ ident ≠ 0 ∨ cursym =
 sigma then goto exit; FB[v]:= cursym;
 FC[v]:= v:= f:= f + 1; cursym:= nextsym; goto again;
exit: if v = w ∧ FA[w] < c4 then
 begin f:= f - 1; realterm:= FA[w]; FA[w]:= 0 end
 else realterm:= w
end;


integer procedure realfactor(ident); integer ident;
begin integer w, v;
 w:= v:= f:= f + 1;
again: FA[v]:= realprimary(ident);
 if cursym ≠ power ∨ ident ≠ 0 ∨ cursym = sigma then goto
 exit; FB[v]:= cursym; FC[v]:= v:= f:= f + 1;
 cursym:= nextsym; goto again;
exit: if v = w ∧ FA[w] < c4 then
 begin f:= f - 1; realfactor:= FA[w]; FA[w]:= 0 end
 else realfactor:= w
end;


integer procedure realprimary(ident); integer ident;
begin integer d, v, l;
 ident:= 0; if cursym = sigma then
 begin realprimary:= 1; goto exit end
 else if cursym = par then
 begin cursym:= nextsym; realprimary:= f:= f + 1;
      FA[f]:= realformula(ident); ER(cursym ≠ rap,
      ‹) missing›); cursym:= nextsym
 end
 else if cursym = point ∨ cursym = lowten ∨ digit then
 realprimary:= NUMBER else if cursym = sum then
 begin realprimary:= f:= f + 1; FA[f]:= sum;
      cursym:= nextsym; ER(cursym ≠ par, ‹(missing›);
      cursym:= nextsym; v:= identifier(true); ER(v = 0,
      ‹identifier missing›); ER(IB[v] ≠ 1,
      ‹wrong identifier›); ER(IF[v] ≥ 1,
     ≮wrong place of identifier≯); IF[v]:= 1; FB[f]:= v;
     ER(cursym ≠ comma, ≮, missing≯); cursym:= nextsym;
     f:= f + 1; FA[f]:= realformula(ident);
     ER(cursym ≠ comma, ≮, missing≯); cursym:= nextsym;
     FB[f]:= realformula(ident); ER(cursym ≠ comma,
     ≮, missing≯); cursym:= nextsym;
     FC[f]:= realformula(ident); ER(cursym ≠ rap,
     ≮) missing≯); cursym:= nextsym; IF[v]:= - 1
end
else if ¬letter then ER(true, ≮wrong symbol≯) else
begin integer identi;
     identi:= ident:= identifier(model); v:= IB[ident];
     if v ≥ 5 then
     begin realprimary:= 1; goto exit end;
     ER(v = 1 ∧ IF[ident] = - 1,
     ≮wrong place of identifier≯);
     if v = 1 ∧ IF[ident] = 0 then IF[ident]:= 2;
     if cursym ≠ sub then
     begin realprimary:= ident; ident:= 0; goto exit end;
     realprimary:= f:= f + 1; FA[f]:= ident; ident:= 0;
     v:= f; FB[v]:= sub; d:= 1;
again: cursym:= nextsym;
     FC[v + d - 1]:= realformula(ident);
     ER(ident ≠ 0 ∨ cursym = sigma,
     ≮identifier or sigma not allowed≯);
     if cursym ≠ comma then goto ex;
     for l:= f step - 1 until v + d do
     begin integer a;
          a:= FA[l];
          FA[l + 1]:= if a < c4 then a else a + 1;
          a:= FB[l];
          FB[l + 1]:= if a < c4 then a else a + 1;
          a:= FC[l];
          FC[l + 1]:= if a < c4 then a else a + 1;
     end;
     FA[v + d]:= FB[v + d]:= 0; f:= f + 1;
     for l:= v step 1 until v + d - 1 do if FC[l] > c4
     then FC[l]:= FC[l] + 1; d:= d + 1; goto again;
ex:  if IC[identi] = 0 then IC[identi]:= d;
     ER(IC[identi] ≠ d, ≮wrong number of subscripts≯);
     ER(cursym ≠ bus, ≮]missing≯); cursym:= nextsym
end;
exit:
end;


integer procedure NUMBER;
begin integer i;
 real val;
 val:= unsignednumber;
```

```
     for i:= c1 + 1 step 1 until num do if RR[i] = val then
     goto exit; i:= num:= num + 1; RR[i]:= val;
exit: NUMBER:= i
end;


real procedure value(g); value g; integer g;
value:= if g < c4 then direct(g) else eval(g);


real procedure prime(g); value g; integer g;
begin integer a, b, c, i, lb, ub;
  real t;
  a:= FA[g]; b:= FB[g]; if a = sum then
  begin i:= b; g:= g + 1; a:= FA[g]; b:= FB[g]; c:= FC[g];
       lb:= - entier( - value(a)); ub:= entier(value(b));
       t:= 0;
       for IF[i]:= lb step 1 until ub do t:= t + value(c)
  end
  else if b = sub then
  begin integer array subs[1:IC[a]];
       b:= IC[a];
       for i:= 1 step 1 until b do subs[i]:= value(FC[g + i
       - 1]); t:= subscripted(a, b, subs)
  end
  else t:= direct(a); prime:= t
end;


real procedure eval(g); value g; integer g;
begin integer a, b, c;
  real t, u;
  a:= FA[g]; t:= if a < c4 then prime(g) else eval(a);
again: b:= FB[g]; c:= FC[g];
  if a = sum ∨ b = sub ∨ c = 0 then goto exit; g:= c;
  a:= FA[g]; u:= if a < c4 then prime(g) else eval(a);
  t:= if b = plus then t + u else if b = min then t - u
  else if b = mult then t × u else if b = div then t / u
  else if b = power then t ↑ u else ⏨600; b:= FB[g];
  c:= FC[g]; goto again;
exit: eval:= t
end;


real procedure direct(g); value g; integer g;
direct:= if g = 0 then 0 else if g = 1 then 1 else if g > c1
then RR[g] else if IB[g] < 2 then IF[g] else if IB[g] = 3
then RR[IE[g]] else if IB[g] = 4 then eval(IE[g]) else ⏨600;
```

```
real procedure subscripted(ident, d, subs); value ident, d;
integer ident, d; integer array subs;
begin integer p;
 p:= count(ID[ident], d, subs); if p > IF[ident] then
 begin PRINTTEXT(⟨undefined⟩); goto exit end;
 p:= p + IE[ident] - 1;
 subscripted:= if IB[ident] = 2 then II[p] else RR[p]
end;


integer procedure count(b, d, subs); value b, d; integer b, d;
integer array subs;
begin integer a, i, t, j, lb, ub;
 integer array save[1:d];
 t:= 0; b:= b - 1;
 for i:= 1 step 1 until d do
 begin b:= b + 3; lb:= - entier( - value(P[b - 1]));
       ub:= entier(value(P[b + 1])); a:= subs[i];
       ER(a < lb ∨ a > ub, ⟨element undefined⟩); ub:= a - 1;
       a:= P[b]; save[i]:= IF[a];
       if i = d then t:= t + ub - lb + 1 else
       for IF[a]:= lb step 1 until ub do t:= t + all(b + 3, i
       + 1, d); IF[a]:= subs[i]
 end;
 for i:= d step - 1 until 1 do
 begin IF[P[b]]:= save[i]; b:= b - 3 end;
 count:= t + 1
end;


integer procedure all(b, i, d); value b, i, d; integer b, i, d;
begin integer a, t, save, lb, ub;
 lb:= - entier( - value(P[b - 1]));
 ub:= entier(value(P[b + 1]));
 if ub < lb then t:= 0 else if i = d then t:= ub - lb + 1
 else
 begin a:= P[b]; save:= IF[a]; t:= 0;
       for IF[a]:= lb step 1 until ub do t:= t + all(b + 3, i
       + 1, d); IF[a]:= save
 end;
 all:= t
end;


procedure readlist(ident, i, t, d, ind, subs, num, COEF);
value ident, i, t, d; integer ident, i, t, d;
integer array ind, subs;
begin integer j, a, b, lb, ub;
 if t = 0 then
 begin lb:= count(ID[ident], d, subs);
```

```
          if lb > IF[ident] then go to exit;
          CUEF[IE[ident] + lb - 1]:= nextnumber; go to ex
    end;
    for j:= 1 step 1 until d do if ind[j] = i then goto fnd;
  fnd: b:= ID[ident] - 1 + 3 × j;
    lb:= - entier( - value(P[b - 1]));
    ub:= entier(value(P[b + 1])); a:= IE[ident];
    for subs[j]:= lb step 1 until ub do
    begin IF[P[b]]:= subs[j];
          if i ≠ t then readlist(ident, i + 1, t, d, ind,
          subs, num, CUEF) else
          begin lb:= count(ID[ident], d, subs);
                if lb > IF[ident] then goto exit;
                CUEF[a + lb - 1]:= nextnumber
          end
    end;
  ex:
  end;


  procedure portions;
  begin integer ident, a, d, b, i, t;
  next: ident:= identifier(true); a:= IB[ident];
    if a ≠ 2 ∧ a ≠ 3 then goto exit; d:= IC[ident];
    if d = 0 then
    begin if a = 2 then IF[ident]:= nextnumber else
          begin num:= num + 1; RR[num]:= nextnumber;
                IE[ident]:= num
          end
    end
    else
    begin integer array ind, subs[1:d];
          if IF[ident] = 0 then
          begin b:= IF[ident]:= all(ID[ident] + 2, 1, d);
                IE[ident]:= if a = 2 then con + 1 else num + 1;
                if a = 2 then con:= con + b else num:= num + b;
          end;
          if cursym ≠ sub then goto exit;
          for i:= 1 step 1 until d do
          begin ind[i]:= 0; cursym:= nextsym;
                if letter then ind[i]:= identifier(true) else
                subs[i]:= nextnumber
          end;
          if cursym ≠ bus then goto exit; cursym:= nextsym;
          t:= 0; if cursym ≠ par then
          begin for i:= 1 step 1 until d do if ind[i] ≠ 0 then
                begin t:= t + 1; ind[i]:= t end
          end
          else
          begin
```

```
        again: cursym:= nextsym; b:= identifier(true);
             t:= t + 1;
             for i:= 1 step 1 until d do if ind[i] = b then
             begin ind[i]:= t; goto fnd end;
             goto exit;
        fnd: if cursym = comma then goto again;
             if cursym ≠ rap then goto exit; cursym:= nextsym
        end;
        b:= ID[ident] - 1;
        for i:= 1 step 1 until d do
        begin b:= b + 3; if ind[i] = 0 then IF[P[b]]:= subs[i]
        end;
        if a = 2 then readlist(ident, 1, t, d, ind, subs,
        con, II) else readlist(ident, 1, t, d, ind, subs,
        num, RR)
   end;
   if letter then goto next
  end;


  real procedure nextnumber;
  begin integer s;
   s:= if cursym = min then - 1 else 1;
   if cursym = plus ∨ cursym = min then cursym:= nextsym;
   nextnumber:= s × unsignednumber
  end;


  integer procedure numvar(type); value type; integer type;
  begin integer i, l, t;
   t:= 0;
   for i:= 200 step 1 until id do if IB[i] = type then
   begin l:= if IC[i] = 0 then 1 else all(ID[i] + 2, 1, IC[i]);
        IF[i]:= l; t:= t + 1
   end;
   numvar:= t
  end;


  integer procedure numcon(type); value type; integer type;
  begin integer i, l, t;
   t:= 0;
   for i:= 1 step 1 until lf do if LB[i] = type then
   begin l:= if LF[i] = 0 then 1 else all(LE[i] + 2, 1, LF[i]);
        LG[i]:= l; t:= t + 1
   end;
   numcon:= t
  end;
```

```
procedure bounds;
begin integer j, i, v, b;
 Boolean fnd;
 for i:= 1 step 1 until lf do
 begin b:= LC[i]; fnd:= false;
      for j:= LA[i] + 1 step 1 until LC[i] do
      begin v:= P[j]; if v = sigma then goto not;
           if v < 200 ∨ v > c1 then go to nextj;
           if IB[v] < 5 then goto nextj;
           if fnd then goto not else fnd:= true;
      nextj:
      end;
      LB[i]:= - LB[i];
 not:
 end;
 for i:= 200 step 1 until id do if IB[i] ≠ 4 then IE[i]:=
 IF[i]:= 0;
end;


procedure pregen;
begin integer i;
 n:= 0;
 for i:= 200 step 1 until id do if IB[i] ≥ 5 then n:= n + 1;
 m:= 0;
 for i:= 1 step 1 until lf do if LB[i] ≥ 0 then m:= m + 1;
 N1:= numvar(5); N:= N1 + numvar(6);
 M1:= numcon(less) + numcon(more); M:= M1 + numcon(equal);
 dmax:= 0;
 for i:= 200 step 1 until id do if IB[i] ≥ 5 ∧ IC[i] > dmax
 then dmax:= IC[i]; RUNOUT; PUNLCR;
 for i:= N1, N, M1, M do FIXP(4, 0, i); PUNLCR; PUNLCR;
end;


procedure generate;
begin integer s, t, i, j, u, h, type, k, l, ident, v, d, lfh, w;
 real sigcoef;
 integer array PV[1:n], PC[1:m], subs[1:1 + dmax];
 real array low, up[1:N], col, rhs[1:M + 1];

 procedure each(b, i, d, proc); value b, i, d;
 integer b, i, d; procedure proc;
 begin integer a, save, lb, ub;
      lb:= - entier( - value(P[b - 1]));
      ub:= entier(value(P[b + 1])); a:= P[b]; save:= IF[a];
      for IF[a]:= lb step 1 until ub do if i < d then
      each(b + 3, i + 1, d, proc) else proc; IF[a]:= save
 end;
```

```
procedure torhs;
begin real a;
      integer g;
      a:= value(k); if type = more then a:= - a; u:= u + 1;
      rhs[u]:= a; NLCR; PRINTTEXT(⪡nr: ⪢); print(u);
      if LF[h] > 0 then PRINTTEXT(⪡subs: ⪢);
      for g:= 1 step 1 until LF[h] do print(subs[g]);
      PUNCH(a)
end;


procedure genbound;
begin integer g, h;
      real a, b;
      a:= value(k); b:= value(P[l - 1]);
      if P[l - 2] = min then b:= - b;
      if abs(b) < ⏨ - 20 then
      begin ER((type = equal ∧ abs(a) ≥ ⏨ - 20) ∨ (type =
            more ∧ a ≥ ⏨ - 20) ∨ (type = less ∧ a ≤ - ⏨ -
            20), ⪡error in bounds⪢); goto out
      end;
      for g:= 1 step 1 until d do subs[g]:= value(P[l + g]);
      h:= if d = 0 then 1 else count(ID[ident], d, subs);
      h:= u + h; a:= a / b; if type = equal then
      begin ER(a < low[h] ∨ a > up[h], ⪡error in bounds⪢);
            low[h]:= up[h]:= a
      end
      else if type = more = b < 0 then
      begin ER(a < low[h], ⪡error in bounds⪢);
            if a < up[h] then up[h]:= a
      end
      else
      begin ER(a > up[h], ⪡error in bounds⪢);
            if a > low[h] then low[h]:= a
      end;
out:
end;


procedure eachvar(b, i, d, proc); value b, i, d;
integer b, i, d; procedure proc;
begin integer a, save, lb, ub;
      lb:= - entier( - value(P[b - 1]));
      ub:= entier(value(P[b + 1])); a:= P[b]; save:= IF[a];
      for IF[a]:= lb step 1 until ub do
      begin subs[i]:= IF[a];
            if i < d then eachvar(b + 3, i + 1, d, proc)
            else proc
      end;
```

```
     IF[a]:= save
end;


procedure some(b, i, d, proc); value b, i, d;
integer b, i, d; procedure proc;
begin integer a, save, lb, ub;
     lb:= - entier( - value(P[b - 1]));
     ub:= entier(value(P[b + 1])); a:= P[b];
     if IG[a] ≠ 0 then
     begin save:= subs[IG[a]];
          if save < lb ∨ save > ub then goto out;
          lb:= ub:= save
     end;
     save:= IF[a];
     for IF[a]:= lb step 1 until ub do if i < d then some(b
     + 3, i + 1, d, proc) else proc; IF[a]:= save;
out:
end;


procedure gencol; comment (ident,d,subs,v);
begin integer i, w, h, lfh, lch, s, type, k, a, l;

     procedure gencoef; comment (lfh,w,h,s,k,type);
     begin integer u, g;
          real sigcoef;

          procedure addcoef; comment (k,sigcoef,type,u);
          begin integer l, g;
               real a;
               for l:= 1 step 1 until d do
               begin a:= value(P[k + l]);
                    if a ≠ subs[l] then got not
               end;
               a:= value(P[k - 1]);
               if P[k - 2] = min then a:= - a;
               a:= a × sigcoef;
               if type = more then a:= - a;
               col[u]:= col[u] + a;
          not:
          end;

          if lfh = 0 then u:= w + 1 else
          begin integer array cs[1:lfh];
               for g:= 1 step 1 until lfh do cs[g]:=
               IF[P[LE[h] - 1 + 3 × g]];
               u:= w + count(LE[h], lfh, cs)
          end;
          if s = 0 then
          begin sigcoef:= 1; addcoef end
          else
          begin sigcoef:= value(P[s - 1]);
                if P[s - 2] = min then sigcoef:= - sigcoef;
                some(s + 3, 1, P[s + 1], addcoef)
          end
     end;
     for i:= 1 step 1 until M + 1 do col[i]:= 0; w:= 0;
     for i:= 1 step 1 until m + 1 do
     begin h:= if i < m then PC[i] else 0; lfh:= LF[h];
          lch:= LC[h]; s:= 0; type:= LB[h];
          for k:= LA[h] + 1 step 1 until lch do
          begin a:= P[k]; if a = sigma then s:= k;
                if a = rap then s:= 0;
                if a ≠ ident then goto nextk;
                for l:= 1 step 1 until d do
                begin a:= P[k + l];
                     if a > c4 + 1 then go to nextl;
                     if a ≥ 200 ∧ a < c1 then
                     begin if IB[a] = 1 then IG[a]:= 1 end
                     else
                     begin if direct(a) ≠ subs[l] then goto
                          nextk
                     end;
                nextl:
                end;
                if lfh = 0 then gencoef else some(LE[h] +
                2, 1, lfh, gencoef);
          nextk: for l:= 200 step 1 until id do
                IG[l]:= 0;
          end;
          w:= w + LC[h]
     end;
     v:= v + 1; NLCR; PRINTTEXT(⟨nr: ⟩); print(v); PUNLCR;
     PUNLCR; FIXP(4, 0, v); if d > 0 then PRINTTEXT(
     ⟨subs: ⟩);
     for i:= 1 step 1 until d do print(subs[i]);
     PUNCH(low[v]); PUNCH(up[v]); PUNLCR;
     for i:= 1 step 1 until M do if col[i] ≠ 0 then
         begin PUNLCR; FIXP(3, 0, i); PUNCH(col[i]) end; PUNLCR;
         FIXP(3, 0, M + 1); PUNCH(col[M + 1]×objective);
end;
s:= 1; t:= n;
for i:= 200 step 1 until id do if IB[i] ≥ 5 then
begin j:= IB[i]; if j = 5 then
     begin PV[s]:= i; s:= s + 1 end
     else
     begin PV[t]:= i; t:= t - 1 end
end;
s:= 1; t:= m;
```

```
      for i:= 1 step 1 until lf do if LB[i] ≥ 0 then
      begin j:= LB[i]; if j ≠ equal then
           begin PC[s]:= i; s:= s + 1 end
           else
           begin PC[t]:= i; t:= t - 1 end
      end;
      u:= 0; rhs[M + 1]:= 0; NEWPAGE;
      for i:= 1 step 1 until m do
      begin h:= PC[i]; type:= LB[h]; NLCR; NLCR; SPACE(20);
           for k:= LD[h] + 1 step 1 until LD[h + 1] do
           PRSYM(string[k]); k:= P[LC[h] + 1];
           if LF[h] = 0 then torhs else eachvar(LE[h] + 2, 1,
           LF[h], torhs)
      end;
      for j:= 1 step 1 until N do
      begin low[j]:= - ⏨600; up[j]:= ⏨600 end;
      for i:= 1 step 1 until lf do if LB[i] < 0 then
      begin type:= - LB[i]; k:= P[LC[i] + 1]; l:= LA[i] + 3;
           ident:= P[l]; d:= IC[ident]; u:= 0;
           for j:= 1 step 1 until n do
           begin h:= PV[j]; if h = ident then goto comp;
                u:= u + IF[h]
           end;
      comp: if LF[i] = 0 then genbound else each(LE[i] +
           2, 1, LF[i], genbound)
      end;
      for j:= 1 step 1 until N do if low[j] = - ⏨600 then
      low[j]:= 0; v:= 0; NEWPAGE;
      for i:= 1 step 1 until n do
      begin ident:= PV[i]; d:= IC[ident]; NLCR; NLCR; SPACE(20);
           for j:= 1 step 1 until c0 do PRSYM(IA[ident,j]);
           if d = 0 then gencol else eachvar(ID[ident] + 2, 1,
           d, gencol)
      end;
      PUNLCR; RUNOUT
     end;

     procedure printstorage;
     begin integer h, i, j;
      PRINTTEXT(‹
identifiers     ›);
      for i:= 200 step 1 until id do
      begin NLCR; print(i);
           for j:= 1 step 1 until 15 do PRSYM(IA[i,j]);
           for h:= IB[i], IC[i], ID[i], IE[i], IF[i] do FIXT(4,
           0, h)
      end;
      PRINTTEXT(‹
program     ›);
      for i:= 1 step 1 until p do
      begin NLCR; print(i); print(P[i]) end;
      PRINTTEXT(‹
linear forms        ›);
```

```
      for i:= 0 step 1 until lf do
      begin NLCR; print(i);
           for h:= LA[i], LB[i], LC[i], LD[i], LE[i], LF[i],
           LG[i] do FIXT(4, 0, h)
      end;
      PRINTTEXT(⟨
strings      ⟩); NLCR;
      for i:= 1 step 1 until str do PRSYM(string[i]); PRINTTEXT(
      ⟨
reals        ⟩); print(num1);
      for i:= c1 + 1 step 1 until num do
      begin NLCR; print(i); print(RR[i]) end;
      PRINTTEXT(⟨
integers     ⟩); print(con1);
      for i:= c1 + 1 step 1 until con do
      begin NLCR; print(i); print(II[i]) end;
      PRINTTEXT(⟨
formulas     ⟩);
      for i:= c4 + 1 step 1 until f do
      begin NLCR; print(i);
           for h:= FA[i], FB[i], FC[i] do print(h)
      end
     end;

struct: start; structuralpart; bounds;
nume: if cursym = INIT then initialization else
     begin ER(cursym = POST, ⟨ post-optimization not implemented⟩);
      ER(cursym = PARA, ⟨ parametrization not implemented⟩);
      ER(cursym ≠ CLOSE, ⟨ wrong symbol ⟩)
     end;
     pregen; if test then printstorage; generate; NEWPAGE;
     if cursym ≠ CLOSE then go to nume;
exit:
end
end
```

Literature

1. Jac.M. Anthonisse

   An input system for linear programming problems,
   part 1: formal description of L.P. problems.

   Report S 371, 21-1-'69,
   Mathematisch Centrum, Amsterdam.