X

CL & CL

Computational Linguistics and Computer Languages

computer
and
automation
institute
hungarian
academy
of sciences

ERRATA

to CL&CL Vol.X.

Page	Line	False	Correct
6	9	"Each command is expended	"Each command is expanded
6	21	sense if trying all	sense of trying all
9	29	These rules are t	These rules are
11	24	must occupy exetly	must occupy exactly
21-22			Swap the two programmes concerned
30	10	in an active rol	in an active role
118			"Table II /continued/" instead of "Table IV"
133	10	eliminated on by	eliminated only by
134			Turn the figure by 90° clockwise

Computer and Automation
Hungarian Academy of Sciences

COMPUTATIONAL LINGUISTICS

AND

COMPUTER LANGUAGES

X.

Budapest, 1975.

Distributor for:

Albania, Bulgaria, China, Cuba, Czechoslovakia, German Democratic Republic, Korean People's Rebublic, Mongolia, Poland, Roumania, U.S.S.R., People's Republic of Viet-Nam, Yugoslavia

K U L T U R A

Hungarian Trading Co. for Books and Newspapers
1389 Budapest,
P.O.B. 149, Hungary

For all other countries:
JOHN BENJAMINS B.V.
Periodical Trade
Amsteldijk 44
AMSTERDAM (Holland)

Responsible Publisher:

Prof Dr VÁMOS Tibor
director of the Computer and Automation
Institute, Hungarian Academy of Sciences

ISBN 963 311 018 1

Országos Müszaki Könyvtár és Dokumentációs Központ házi sokszorositója, F.v.: Janoch Gyula

CONTENTS

BACKTRACK FORTRAN IMPLEMENTED WITH THE HELP OF THE MACRO PROCESSOR MP/O

Julianna FABÓK
Computer and Automation Institute,
Hungarian Academy of Sciences
Budapest, Hungary

INTRODUCTION

Backtrack programming is a technique useful in writing programs "for solving problems expressible as a set of possible alternatives, called a goal tree, where not all of the alternatives will lead to the desired goal. At each branching point in the tree, a decision must be made as to which alternative to try next". /D.C.Smith and H.J.Enea/ If a wrong branch is tried, the forward searching fails, and the program returns to the previous decision point and selects another alternative.

The first exact theoretical formulation of backtrack programming was made by Golomb,S.W. and Baumert, L.D. in 1965. Since then a lot of languages with backtrack features has been implemented. Some of them are created by adding some backtrack instructions to the existing languages, others are totally new ones.

There are two different aspects of backtrack programming. They are different from each other in the method of realization of the return to the previous decision point. One of them is the "sequential backtracking". In this case returning is made step by step, statement by statement, by undoing all the effects of the actual statement. Reaching the previous decision point, everything will be reset to its original condition. The other aspect of backtrack programming is "<u>state backtracking</u>".

It means that in every decision point the state of the machine is saved /the current values of all variables including the system variables/. When a failure occurs, the state of the machine in the previous decision point is restored.

Our work is in close connection with the sequential aspect of backtracking. The fundamental work of Floyd,R. was presented in 1967. He says: "Each command is expended into one or more commands, some of which carry out the effect of the original command in the nondeterministic algorithm, and which also stack information required to reserve the effect of the command when backtracking is needed, while others carry out backtracking by undoing all the effects of the first set". He also presented the transformation rules, how to convert a flowchart describing a backtrack algorithm into a conventional one.

We propose to use the phrase "backtrack algorithm" instead of the phrase "nondeterministic algorithm", since the backtracking algorithms are nondeterministic in the sense of having "free will", but they are deterministic in the sense if trying all the possible alternatives.

Cohen, J. and Carton,E. extended the FORTRAN language by adding some backtrack instructions to it. They made a syntax-directed translator, which converts a program written in backtrack FORTRAN into another one, written in standard FORTRAN. They presented the exact description of the syntax of this extension in BNF.

The same backtrack instructions are used by us, but instead of writing a translator, a macro processor was used to implement the backtrack FORTRAN. The macro processor MP/O has been developed in our Institute.

In this paper we survey the transformation rules by Floyd and the exact specification of the adopted backtrack instructions.

The most important characteristics of the macro processor MP/O and some problems of the implementation will be discussed, too.

Finally an example will be presented, how to use the backtrack FORTRAN to solve problems in the field of artificial intelligence.

1. FLOYD'S TRANSFORMATION RULES

Backtrack languages are means "to simplify the design of a backtracking algorithm by allowing considerations of program book-keeping required for backtracking to be ignored" /Floyd/ To do this we use stacks and flags. More exactly we use one flag T and three stacks: M /memory/, W /write/, R /read/. The stack M is used to store the values of the variables as the process of the execution is going on. The stack W contains the components of the result. It is printed only if a successful termination is reached. The stack R serves to preserve the input data. Two pointers "max" and "min" are introduced. The "max" pointer keeps track of the last element actually read, the "min" pointer points to the element that should be considered when the next backtrack read command is activated. The flag T is used to make the paths of execution different in case of fork.

Fig.1 shows the transformation rules.

Only a few of them need explanations, the others are self-evident.

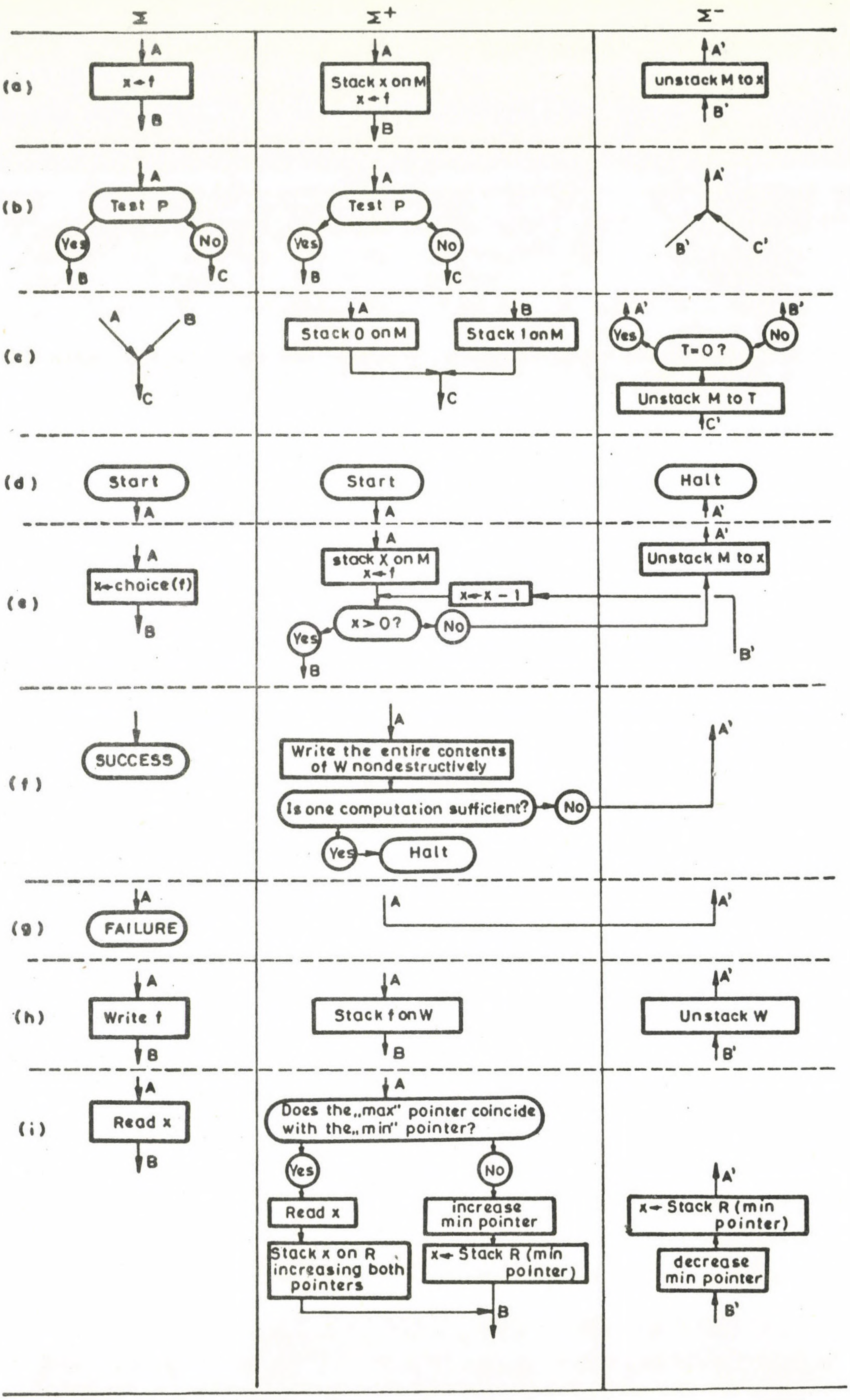

Σ
Σ⁺
Σ⁻
(a)
x ← f
Stack x on M
x ← f
unstack M to x
(b)
Test P
Yes
No
(c)
Stack 0 on M
Stack 1 on M
T = 0 ?
Unstack M to T
(d)
Start
Halt
(e)
x ← choice(f)
stack X on M
x ← f
x ← x − 1
x > 0 ?
Unstack M to x
(f)
SUCCESS
Write the entire contents of W nondestructively
Is one computation sufficient?
Halt
(g)
FAILURE
(h)
Write f
Stack f on W
Unstack W
(i)
Read x
Does the „max" pointer coincide with the „min" pointer?
Read x
Stack x on R increasing both pointers
increase min pointer
x ← Stack R (min pointer)
x ← Stack R (min pointer)
decrease min pointer

Fig. 1.

In the case of standard instructions /assignment (a), conditional branch (b), fork (c), start (d)/ we need tools to preserve and to reset the former values of the variables and to keep track the path of the execution. The new backtrack command units are CHOICE, SUCCESS and FAILURE as well as backtrack read and backtrack write. The X=CHOICE (f) command (e) means that after the previous value of X has been stacked, the forward part is executed with value X=f. If backtracking is needed it is repeated with a decreased value of X (X=X-1) until it reaches the value 0. In the latter case, after the original value of X has been restored, backtracking continues to the command which precedes the CHOICE command. The CHOICE command is the most important backtrack command. With its help you can try all the possible alternatives at a decision point, if X is the serial number /or index/ of the alternatives.

The SUCCESS command (f) results the contents of the stack W to be printed. The programmer has two possibilities: his program either stops or proceeds in its backtracking.

The FAILURE command (g) shows that backtracking is necessary. In the case of backtrack write (h) the value of an output variable is stacked in W in the forward part and unstacked in the backtracking one.

The most complicated command is the backtrack read (i). We have mentioned before the function of the pointers "max" and "min". If they coincide with each other, a real read operation is needed.

Otherwise we can get the desirable value from the stack R /"min" pointer/. When backtracking occurs the previously read value will be restored from the stack R. These rules are t totally mechanical, so it is very easy to add them to an existing programming language.

2. THE SYNTAX OF THE ADOPTED BACKTRACK INSTRUCTION

In this part, the exact syntax of the instructions allowed in backtrack FORTRAN is described in BNF. This specification was published by Cohen and Carton in 1974. It is as follows:

```
<backtrack program>::=
          <sequence of normal instructions>
          START
          <sequence of standard or backtrack instructions>
          END
<backtrack instructions>::=<choice>|<success>|<failure>|
          |<backtrack write>|<backtrack read>
<choice>::= <variable>=CHOICE(<arithm.expr>)
<success>::= SUCCESS|SUCCESS QUIT
<failure>::= FAILURE
<backtrack write>::= OUT<variable>|OUT<constant>
<backtrack read>::=  IN<variable>|IN<constant>
<standard instruction>::=<assignment>|<go-to>|<if>
<assignment>::=<variable>=<arithm.expr.><inverse>
<inverse>::= INV<arithm.expr.>|INV NIL|empty
<go-to>::= GOTO<label>
<if>::= IF(<boolean expr.>)<simple statement>
<simple statement>::=<go-to>|<success>|<failure>
```

Any <standard instruction> or <backtrack instruction> may be preceded by a FORTRAN numeric <label>. A <variable> can be a subscripted or simple variable.

The undefined concepts are used with their usual meanings /arithm. expr., boolean expr., variable, constant, label, etc./.

All the variables are assumed to be INTEGER.

The instruction SUCCESS QUIT must be used if we are intrested in finding only one solution. Using the instruction SUCCESS, we shall find all the existing solutions.

In this implementation we have means to forbid stacking if it is not necessary. We may write INV NIL after an assignment command to order the value of the variable not to be stacked and unstacked at all. Writing an arithmetic expression after INV, the value of the variable will not be stacked in the forward part of execution, but the value of the arithm. expression will be assigned to the variable in the backtracking direction.

3. THE MOST IMPORTANT CHARACTERISTICS OF THE MACRO PROCESSOR MP/O

Sequential backtrack is specially suitable to be implemented with the help of a macro processor, for it can be defined exactly how a backtrack command expands to one or more instructions. The macro processor MP/O has been developed by our colleague, FARKAS,E. with the aim of language extension and language translation.

Here we want to survey its most important features which are particularly suitable for our purposes.

3.1 The MP/O is a text macro processor. It means that outside of macro calls no syntactic analysis of the source text is performed.

3.2 It works on larger text units. A unit is one line. Patterns and macro calls must occupy exetly one line of the text. One line in the source text is replaced by one or more lines.

3.3 The method of identifying the macros to be expanded is pattern matching. That is, each macro has been associated with a pattern which consists of a sequence of fixed strings, so-called "keywords", interspersed with arbitrary strings, i.e. parameters. A macro is identified by the occurrence of its pattern. This feature is very useful for us for the recognition of the backtrack FORTRAN commands is totally automatic. We must only specify the macro bodies.

3.4 Macro variables can be used. A variable with the value of type INTEGER belongs to every letter of the alphabet. We have also facilities to perform certain restricted macro-time arithmetic with these variables. They are useful for generating constants, for example FORTRAN numeric labels and other references and as flags for switches.

3.5 Nested macros can be used. Both macro definitions and macro calls can be included in a macro definition. These are evaluated only at the call of the outer macro.

3.6 Every macro has a macro number and a successor. It means that macro definitions form one or more chains, which are ordered sets of macro definitions. These chains may be linked to one an other, that is one or more macros can have the same successor.

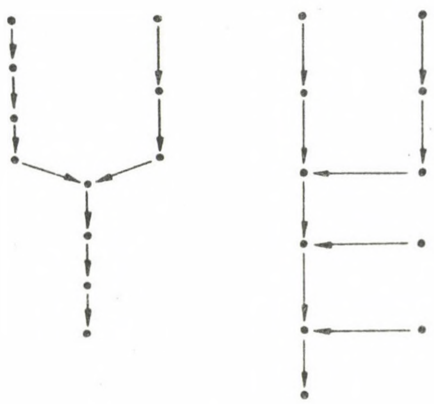


Fig 2

A macro call being evaluated, it will be compared only with the patterns belonging to the chain which is actually assigned. It is a tool to make the process of evaluating more efficient.

3.7 The macro processor MP/O has <u>macro-time facilities</u>, too. They serve for giving instructions to the macro processor itself. As a result of their effects, the inner state of the macro processor will be changed. The macro-time facilities of the MP/O are:

3.7.1 <u>Macro-time variables</u>. We have mentioned them before. The restricted arithmetic operations are:

+	addition
-	substraction
*	multiplacation
/	integer division
:	remaindering

An arithmetic expression may consist of at most two variables with an operator between them. Of course, a constant may stand instead of any variable

3.7.2 <u>Macro definitions</u>. They have four important parts. The <u>macro head</u> contains the macro number and the number of the successor. /first line/

The <u>pattern</u> consists of the keywords and the formal parameters. /second line/

The <u>body</u> is the text to be copied.

<u>Macro tail</u>: END OF MACRO. /last line/
It is only the body that may contain macro calls or macro-time statements.

3.7.3 Statements for <u>control of matching</u>. They are tools to assign the chain of macro definitions which takes part in the process of matching. /see 3.6/

There are two types of them. A chain may be assigned permanently by the command BEGIN AT.
The command MATCH WITH results a temporary assignment which is valid during the evaluation of only one line.

3.7.4 Statements to transfer control. They give possibilities to skip one or more lines. There are conditional and unconditional SKIP commands. The condition is the value of the given macro variable being positive, zero or negative.

3.7.5 Statements to assign the input device

4. SOME REMARKS ON THE IMPLEMENTATION

This part deals with the most interesting problems of how to use this macro processor to implement backtrack FORTRAN. We shall discuss three important problems, which are:

the question of memorization
the generation of reference numbers
the use of the chains of macro definitions.

4.1 The macro processor "remembers"

We can formulate the problem of the implementation as follows:
A backtrack instruction has to be replaced by a forward and a backward part, but they are not next to each other. The structure of the generated new program can be seen in Fig.3.

backtrack program	the generated program
K1	K1+
K2	K2+
K3	K3+
	K3-
	K2-
	K1-

Where K1, K2, K3 are backtrack commands
K1+, K2+, K3+ are their forward parts in order
K1-, K2-, K3- are their backward parts in order

Fig.3

The macro processor can directly generate the forward parts. Our task is to provide for the memorization of backward parts by the macro processor. We do this with the help of macro definitions. On processing every backtrack instruction, we define a back macro containing the backward part of this command and the macro call of the back macro defined on processing the previous backtrack instruction. The first phase of the work of the macro processor is to evaluate the lines of the source text line by line, to generate their forward parts and to define the corresponding back macros if it is necessary. The lines which are not backtrack commands will only be copied without any modification. The macro calls of the back macros defined in the first phase will take place only after the processing of the last line of the backtrack program /END/. This is the second phase of the work of the macro processor. In this phase the nested back macro calls are evaluated.

The number of the levels is equal to the number of the backtrack commands in the program.

For example we present the macro corresponding to the backtrack instruction <variable>=<arithm.expr.> INV

<arithm.expr.>. The question-marks in the pattern mark the places of the parameters. These parameters are referred in the body by their serial number after an upward arrow. The macro variables are referred similarly. The macro-time statements begin with the "warning" mark:&. The pattern of every back macro is the same /()/, but their macro numbers are different and they are linked to the same chain of macro definitions. The macro variable Q contains the serial number of the actual back macro. It increases in the first phase of processing and decreases during the second one.

macro definition	explanations
&MACRO NO 7 NEXT 10	macro head; macro number: 7, the macro number of the successor: 10;
?=?INV?	pattern;
&MATCH WITH 1 } ↑1=↑2	forward part without modification;
P=↑Q+1	P is an auxiliary variable;
MACRO NO ↑P NEXT ↑Q	generating a new back macro definition with increased macro number;
?()	common pattern for back macros;
&MATCH WITH 1 } ↑1=↑3	backward part of this command;
&Q=↑Q-1	the value of Q decreases;
&MATCH WITH ↑Q ()	it calls the previous back macro;
&END OF MACRO	back macro ends;
&Q=↑Q+1	the value of Q increases;
&END OF MACRO	the whole macro ends;

4.2 The generation of reference numbers

Sometimes we must set up connections between the forward and backward parts of an instruction which are far away from each other. Macro variables are used to do this. There are two important ways of using them. We will call them bound and unbound usage.

4.2.1 The unbound usage is when both, the referred label and the reference are generated by the macro processor. In this case:

- the label may be an arbitrary number
- it must not occur so far.

To satisfy these conditions we use a macro variable /Y/ pointing to the value of the actual numeric label. It decreases from a fixed value X during the first phase of processing and decreases in the second one.
For example we present the macro corresponding to the command FAILURE;

&macro definition	explanations
MACRO NO 10 NEXT 5	macro head; macro number: 10, macro number of the successor: 5;
?FAILURE	pattern;
&MATCH WITH 1 GOTO↑Y	forward part;
&P=↑Q+1	auxiliary variable;
&MACRO NO ↑P NEXT ↑Q	back macro definition begins;
?()	common pattern for back macros;
&Y=↑Y+1 &Q=↑Q-1	macro variables are modified;

&macro definition	explanations
&MATCH WITH 1 ↑Y CONTINUE	backward part;
&MATCH WITH ↑Q ()	call of the previous back macro;
&END OF MACRO	back macro ends;
&Y=↑Y-1 &Q=↑Q+1	macro variables are modified
&END OF MACRO	the whole macro ends;

4.2.2 The bound usage is when either the reference or the referred label are fixed by the source text. The simplest example is the GOTO statement.

backtrack program

```
      .
  GOTO N
      .
      .
      .
      .
  N CONTINUE
      .
      .
      .
      .
```

generated program

```
          .
  STACK 1 ON M          )
  GOTO N                )
          .             )
          .             )  forward part
          .             )
  STACK O ON M          )
N CONTINUE              )
          .             )
          .             )
          .
  UNSTACK M TO T        )
  IF T.EQ.1 GOTO X+N    )
          .             )
          .             )  backward part
          .             )
X+N CONTINUE            )
          .             )
```

In this situation the backward part of the command N CONTINUE refers to the backpart of the command GOTO N. But we do not know where the statement GOTO N and its backpart are. For this reason we mark the backpart of the command GOTO N with a label with value X+N. X is a fixed value, especially it may be the same as in the previous part.

It results that you must not jump twice or more times to the same label, because in this case a label would occur twice or more times in the generated program. It is a disadvantage of our work, but it can be eliminated by inserting a new line in the backtrack program: new label CONTINUE;

4.3 The use of the chains of macro definitions

The chains of macro definitions are to make the work of the macro processor more efficient. We use the following chains in this work: /see Fig.4/

START	type of backtrack commands	stack instructions	back macros

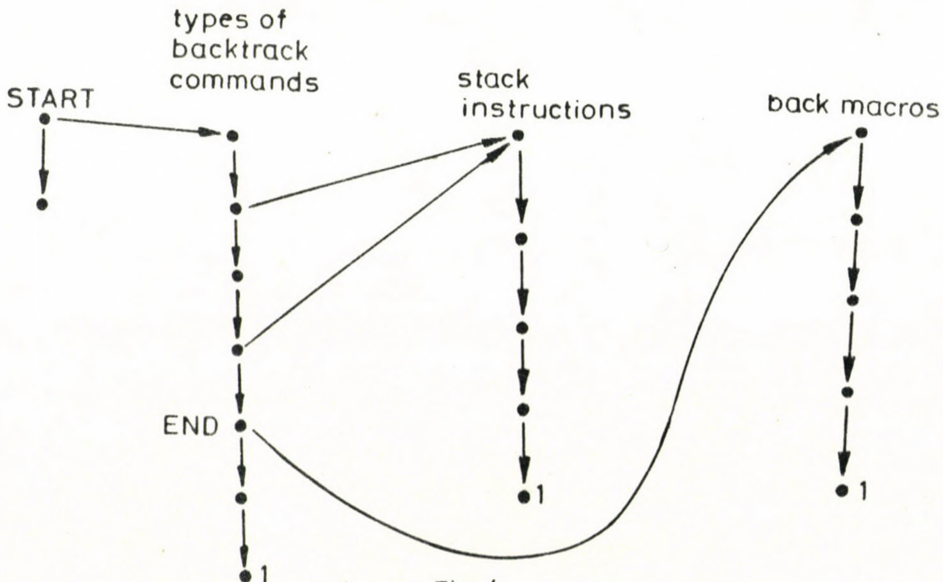


Fig.4.

- before reaching the command START we look only for this instruction,
- after this, the patterns of the backtrack commands are scanned,
- if the processing of a backtrack instruction uses stacks, the stack and unstack macros are a different chain,
- after the processing of the command END, the chain of back macro definitions will be activated.

Finally we want to present the restrictions connected with the specifications of the macro processor MP/O.

1. It is not allowed to jump twice or more times to the same label. /We have mentioned this before./

2. The programmer may not use the total scope of the FORTRAN numeric labels, because the macro processor reserves an interval of possible labels as its own.

3. Spaces are significant inside the backtrack commands.

4. Three array-names and four variable-names are reserves. These are:

array	variable
stack M	pointer for M
stack W	pointer for W
stack R	pointer for R
	flag T

5. EXAMPLE: THE PROBLEM OF THE EIGHT QUEENS

The classical example for backtrack algorithm is the eight queens' problem. This problem consists of placing eight queens on a chessboard so that no two attack, i.e. there is only one queen in each row, column, or diagonal of the board. In the program we take advantage of the fact that the sum and difference of the row number and column number of an element in a diagonal is constant. In this part a backtrack FORTRAN program for solving this problem and the generated normal FORTRAN one are presented.

The generated normal FORTRAN program

```
1       MAIN
        DIMENSION IA(8), IB(15), IC(15)
        DO 1 I=1,8
1       IA(I)=Ø
        DO 2 I=1,15
        IB(I)=Ø
2       IC(I)=Ø
        IROW=Ø
        ICOL=1
         IRPC=Ø
         IRMC=Ø
        DIMENSION IZM (5ØØ)
         IZPTM=1
         IZPTW=4Ø1
           IZM(IZPTM)=Ø
        IZPTM=IZPTM+1
3       CONTINUE
           IZM(IZPTM)=    IROW
        IZPTM=IZPTM+1
        IROW=8+1
999                IROW=    IROW-1
        IF(IROW-Ø) 2ØØØ, 998, 2ØØØ
2ØØØ     CONTINUE
           IZM(IZPTM)=    IRPC
        IZPTM=IZPTM+1
        IRPC=IROW+ICOL-1
           IZM(IZPTM)=    IRMC
        IZPTM=IZPTM+1
        IRMC=IROW-ICOL+8
        IF((IA(IROW)+IB(IRPC)+IC(IRMC))-1)2ØØ1, 997, 997
2ØØ1     CONTINUE
        IA(IROW)=1
        IB(IRPC)=1
        IC(IRMC)=1
        IZM(IZPRW)=       IROW
        IZPTW=IZPTW+1
        IF(ICOL-8) 2ØØ2, 996, 2ØØ2
2ØØ2     CONTINUE
        GOTO 995
996     IZPTW=IZPTW-1
        WRITE(1Ø,994)(IZM(I), I=4Ø1, IZPTW)
994     FORMAT(1HX,8I3)
        IZPTW=IZPTW+1
        GOTO 993
```

```
      995   CONTINUE
           ICOL=ICOL+1
              IZM(IZPTM)=1
           IZPTM=IZPTM+1
           GOTO 3
      1ØØ3  CONTINUE
           ICOL=ICOL-1
            CONTINUE
           IZPTW=IZPTW-1
            IROW=IZM(IZPTW)
           IC(IRMC)=Ø
           IB(IRPC)=Ø
           IA(IROW)=Ø
      997   CONTINUE
              IZPTM=IZPTM-1
                 IRMC=IZM(IZPTM
              IZPTM=IZPTM-1
                 IRPC=IZM(IZPTM)
           GOTO 999
      998     IZPTM=IZPTM-1
                 IROW=IZM IZPTM
              IZPTM=IZPTM-1
           IFLAG=IZM(IZPTM)
           IF(IFLAG-1) 2ØØ3, 1ØØ3, 2ØØ3
      2ØØ3  CONTINUE
           STOP
           E N D
           F I N I S H
```

Backtrack FORTRAN program for the eight queens problem

```
           MAIN
           DIMENSION IA(8), IB(15), IC(15)
           DO 1 I=1,8
      1    IA(I)=Ø
           DO 2 I=1,15
           IB(I)=Ø
      2    IC(I)=Ø
           IROW=Ø
           ICOL=1
            IRPC=Ø
            IRMC=Ø
           START
      3    IROW=CHOICE(8)
           IRPC=IROW+ICOL-1
           IRMC=IROW-ICOL+8
           IF((IA(IROW)+IB(IRPC)+IC(IRMC)).GE.1)FAILURE
           IA(IROW)=1 INV Ø
           IB(IRPC)=1 INV Ø
           IC(IRMC)=1 INV Ø
           OUT IROW
           IF(ICOL.EQ.8)SUCCESS
           ICOL=ICOL+1 INV ICOL-1
           GOTO 3
           END
```

To understand the program we note that we represent the chessboard by three one-dimensional arrays IA(8), IB(15), IC(15). A "one" in IA(I), IB(J),IC(K) indicates that row I, left diagonal J and right diagonal K are occupied. To place a queen in the row I and column K means:

```
IA(I)= 1
IB(I+K-1)=1
IC(I-K+8)=1   .
```

To remove this queen means to change this values into zero. The result is represented by an eight dimensional vector. The value of the component I shows that which row in the column I a queen has been placed in.

The algorithm is as follows:

1. we consider the first column;
2. a row is chosen form 8 to 1;
3. test is executed if a queen can be placed in the current column and row.
   If so, go to step 4,
   otherwise backtrack /removing the queen go back to step 2/;
4. the queen is placed in the current row and column position and the row number is stored for the result;
5. test takes place if all queens have been placed.
   If so, the result is printed and the program stops,
   otherwise the next column is chosen and go to step 2.

REFERENCES

[1] Baumert,L.D. and Golomb,S.W.: Backtrack programming
J.ACM Vol 12, No 4, /Oct.,1965/ 516-524.

[2] Floyd,R.W.: Nondeterministic Algorithms
J.ACM Vol 14, No 4 /Oct., 1967/ 636-644

[3] Cohen, J. and Carton,E.: Non-deterministic FORTRAN
The Computer Journal Vol 17, No 1 /Febr., 1974/ 44-51.

[4] Smith,D.C. and Enea,H.J.: Backtracking in MLISP2.
Third International Joint Conference on Artificial Intelligence 1973, 677-685.

[5] Farkas,E.: Az MP/O makroprocesszor.
MTA SzTAKI"Tanulmányok" /"Reports" of the Computer and Automation Institute, Hung.Ac. Sci., Budapest, 12/1973., 21-48.

A COMPILER ORIENTED SYNTAX DEFINITION

Ernő FARKAS
Computer and Automation Institute,
Hungarian Academy of Sciences
Budapest, Hungary

It is well-known that the meta-language was a very important discovery on the way of the more precise description of programming languages, and of the development of common translation technics. However, it is well-known too, that the meta-language is not suitable for each language and even in the languages described well by the meta-language there are parts of the syntax which are out of the definition, for example: if there is an array in the program declared as two dimensional and we use it with three indexes then most of the compilers send an error message, however, this fact may not be established on the basis of the meta-language.

Here in after we want to give a syntax definition based on the meta-language, although the definitional rules are also taken into consideration. Here the "definitional" attribute is used in a very wide sense. The scheme written below makes it possible to examine such properties of the program which were earlier considered as a part of the semantics or a tool of the program debugging. For example, we may check whether an index variable of a cycle is modified inside the cycle, or the fact that in a part of the program which variable can get a value and so on. So we have the possibility to send one error message for one error, in that point where the mistake is most striking. This type of the definition does not mean a new type translation technic but this step allows to get a higher compatibility between the different implementations of the language:

1. We have the possibility to decide more precisely, which kind of program is correct formally and which is erroneous.
2. What kind of errors are required to be detected in the level of translation /and what kind of error in the running/.
3. It is possible to create a uniform error diagnostic system for a language.

THE SYNTAX DEFINITION

Let be "A" the set of the permitted symbols of the language, and we will denote with "A^*" the set of the finite strings from the elements of A.

"A language is a set of such strings from A^* which are corresponded to prototypes" derived by a meta language.

$L \subset A^*$ will be a language if it fulfils the definition below:

Let be the triplet $\langle B,s,\gamma \rangle$ a meta-language, where $B=T \cup N$ and $T \cap N = \emptyset$. T is the set of the terminal symbols and N is the set of the nonterminal symbols.
$s \in N$ is the beginning symbol.
γ is a finite set of substituting rules, in the form $n \to x$, where $n \in N$ and $x \in (T \cup N)^*$.
Let be further $T=A \cup E$ and $A \cap E = \emptyset$, where A is the set of permitted symbols, as above; and E is the set of so-called elementary objects. Hence

$$A \subset T \subset B \ .$$

Let be given in addition an infinite enumerable set, V /the set of the states of the vocabulary/ and v_0 its special element the beginning state. Let be $F \subset V$ the set of the legal final states.

At the end, let be $g \in \{(A x V x E) \rightarrow V\}$ a partial function the so-called vocabulary function.

Let $\ell \in L \subset A^{*}$, if and only if there exists a partition of

$$\ell = x_1, x_2, \ldots x_n \qquad (x_i \in A^{*})$$

so that there exsists such a $t \in T^{*}$ which can be derived from s by the rules of the meta-language, /in the usual way/, and

$$t = y_1, y_2, \ldots y_n \qquad (y_i \in T^{*})$$

and "ℓ" match to "t" in the sense:

$$\begin{aligned} &\text{if } y_i \in A^{*} \quad \text{then } x_i = y_i \\ &\text{else } y_j \in E \text{ and} \quad g(x_{j1}, v_o, y_{j1}) = v_{j1} \\ &\qquad\qquad\qquad\qquad g(x_{j2}, v_{j1}, y_{j2}) = v_{j2} \\ &\qquad\qquad\qquad\qquad \vdots \\ &\qquad\qquad\qquad\qquad g(x_{jk}, v_{jk-1}, y_{jk}) = v_{jk} \end{aligned}$$

for all $y_j \in E$, and $v_{jk} \in F$.

This means informally:

By means of the meta-language we are forming a tree structure On the leaves of the tree, there are either strings from A^{*} /key words/ or elementary objects /labels, variables, etc./. It must be an one-one correspondence between the key words in the tree and the key words in the object language. If there is an elementary object on the leaf of the tree, we have to decide by the function "g" whether the corresponding string "a" is compatible with the elementary object "e" and with the present state "v" of the vocabulary. If it is so, then we can go on, and the vocabulary gets a new state. If they are not compatible, we have several ways for sending error messages and it is advisible to define also these ways at the forming of the syntax. Finally, the vocabulary must have a state in which all the references are satisfied, i.e. in the program there may not occur any object or attribute of an object which was referred but not established.

The vocabulary function is shown in the Appendix in a rather tedious example.

APPENDIX

Let us suppose that we have a language, which is very close to the FORTRAN II. /The FORMAT, EQUIVALENCE, COMMON instructions are not involved into the example, but they may be realized without any difficulties. The only restriction is that the label at the end of a cycle must be the label of a CONTINUE instruction./ It is important for the fact that no elementary objects of the body of the cycle may appear in the program after that point where the label indicates the end of the cycle.

Let us have a small program:

```
      DIMENSION X(50)
      READ K
      DO 110 I=1,K
      READ X(I)
110   CONTINUE
      Y=0
      Z=0
      DO 120 I=1,K
      IF(X(I))111,120,112
111   Y=Y+X(I)
      GO TO 120
112   Z=Z+X(I)
120   CONTINUE
      WRITE Y,Z
      STOP
      END
```

And let us suppose that we are able to derive by the meta-language the string:

```
        DIMENSION e1(e2)
        READ e8
        DO e3     e5=e6,e6
        READ e15(e10)
e7      CONTINUE
        e13=e10
        e13=e10
        IF(e15(e10)) e4,e4,e4
e7      e13=e14+e15(e10)
        GOTO e4
e7      e13=e14+e15(e10)
e7      CONTINUE
        WRITE e12,e12
        STOP
        END
```

Where the elementary objects mean:

e_1	array in declaration
e_2	integer number
e_3	reference for a label in a DO instruction
e_4	reference for a label in a jump instruction
e_5	index variable of a DO cycle
e_6	parameter of a DO cycle
e_7	label
e_8	integer variable
e_9	integer variable which get value
e_{10}	integer value /variable or number/
e_{11}	integer array
e_{12}	real variable
e_{13}	real variable which get value
e_{14}	real value /variable or number/
e_{15}	real array

The vocabulary V is formed as a pairlist, i.e. a list of sublists where the head /CAR/ of the sublists is an element and the tail /CDR/ is its attributes.

The attributes are:

VARI	variable
NUMB	number
INT	integer
REAL	real
ARRAY	array
CLOSED	may not use it in an active rol
DO	the label of a non complete DO cycle
EXIST	existing label

The vocabulary has a final state if all the labels in it are existing.

Figure 1 shows the TRANS function which is the vocabulary function defined in pure Lisp. Figure 2 swhows the states of the vocabulary during the checking of the current program.

The program, has been executed by the R10 minicomputer in a 16K byte version of the Lisp interpreter.

Figure 1.

```
(DEFINE(QUOTE((TRANS (LAMBDA(E,X,V)
(COND

((EQ Q E1)(COND
((FIND X V) ERROR)
(I (CONS (LIST X (INT X),*ARRAY*) V))))

((EQ E E2)(COND
((AND (IN *INT*(GET X,V))(IN *NUMB*(GET X,V)))(UPDATE(GET X,V)V))
(T ERROR) ))

((EQ E E3)(COND
((FIND X V) (COND
((IN *DO*(FIND X V))(CONS(LIST X,*DO*)V))
(T ERROR) ))
(T (CONS(LIST X*DO*) V)) ))

((EQ E E4)(COND
((NOT (FIND X V))(CONS(LIST X) V))
((IN *CLOSED*(FIND X V)) ERROR)
(T V) ))

((EQ E E5)(COND
((AND(AND(IN *VARI* (GET X,V))(IN *INT*(GET X,V)))
               (NOT (IN *CLOSED* (GET X,V))))
(CONS(TAIL (CAR V)X)(UPDATE
(TAIL (GET X V) *CLOSED*)(CDR V))))
(T ERROR) ))

((EQ E E6)(COND
((IN *INT* (GET X V))(COND
((IN *VARI* (GET X V))(CONS (TAIL(CAR V) X)(UPDATE(TAIL
               (GET X V)*CLOSED*)(CDR V))))
((IN *NUMB* (GET X V))(CONS(CAR V)(UPDATE(GET X,V)(CDR V))))
(T ERROR)))
(T ERROR)))

((EQ E E7)(COND
((NOT (FIND X,V))(CONS (LIST X, *EXIST*)V))
((IN *DO* (FIND X,V))(CLOSE X,V))
((IN *EXIST*(FIND X,V)) ERROR)
(T (CHEK X,V))))

((EQ E E8)(COND
((AND (IN *INT* (GET X V))(IN *VARI*(GET X V)))(UPDATE
               (GET X V)V))
(T ERROR)))

((EQ E E9)(COND
((AND(AND(IN *INT* (GET X,V))(IN *VARI*(GET X,V)))
               (NOT(IN *CLOSED*(GET X,V))))(UPDATE (GET X V)V))
(T ERROR)))

((EQ E E 1Ø)(COND
((AND(IN*INT*(GET X,V))(NOT(IN*ARRAY*(GET X,V))))(UPDATE
               (GET X V)V))
(T ERROR) ))
```

```
((EQ E E11)(COND
((AND (IN *INT*(FIND X V))(IN *ARRAY*(FIND X V))) V)
(T ERROR)))

((EQ E E12)(COND
((AND (IN *REAL* (GET X V))(IN *VARI*(GET X V)))(UPDATE
                  (GET X V)V))
(T ERROR)))

((EQ E E13)(COND
((AND(AND(IN*REAL*(GET X,V))(IN*VARI*(GET X,V)))
                  (NOT(IN*CLOSED*(GET X,V))))(UPDATE (GET X V)V))
(T ERROR)))

((EQ E E14)(COND
((AND(IN *REAL*(GET X,V))(NOT(IN *ARRAY*(GET X,V))))(UPDATE
                  (GET X V)V))
(T ERROR) ))

((EQ E E15)(COND
((AND (IN*REAL*(FIND X V))(IN *ARRAY*(FIND X V))) V)
(T ERROR)))

(T ERROR2)
))))))
(DEFINE(QUOTE(

(CLOSE (LAMBDA (X,V)(COND
((NOT (FIND X,V))V)
((EQ X(CAR(CAR V)))(OPEN(CDR(CDR(FIND X,V)))(CONS
                  (LIST X,*EXIST*,*CLOSED*)(CLOSE X (CDR V)))))
((IN*EXIST*(CAR V)) (CONS(TAIL (CAR V),*CLOSED*)
                  (CLOSE X (CDR V))))
(T(CONS(CAR V)(CLOSE X (CDR V)))) )))

(CHEK(LAMBDA (X,V)(COND
((IN*DO*(CAR V)) ERROR)
((EQ X (CAR(CAR V)))(CONS(LIST X,*EXIST*)(CDR V)))
(T(CONS(CAR V)(CHEK X,(CDR V)))) )))

(CURTAIL (LAMBDA (X) (COND
((EQ(CDR Y)NIL)NIL)
(T(CONS(CAR Y)(CURTAIL(CDR Y)))) )))

(OPEN(LAMBDA(Y,V)(COND
((NULL Y) V)
(T(OPEN(CDR Y)(UPDATE(CURTAIL(FIND(CAR Y) V)) V))) )))

(TAIL (LAMBDA (S,Y)(COND
((NULL S)(LIST Y))
(T(CONS(CAR S)(TAIL(CDR S)Y))) )))

(GET (LAMDA (X,V)(COND
((FIND X V)(FIND X V))
(T (LIST X (INT X)(VARI X))))))

(IN(LAMBDA (P,L) (COND
((NULL L) NIL)
((EQ (CAR L) P) T)
(T(IN P (CDR L))))))
```

```
(UPDATE (LAMBDA (L V)(COND
((NOT(FIND (CAR L) V))(CONS L V))
(T (COND
 ((EQ(CAR L) (CAR(CAR V)))(CONS L (CDR V)))
 (T (CONS(CAR V)(UPDATE L (CDR V))))
    )) )))

)))
```

Figure 2.

```
VØ NIL

V1=TRANS[E1;X;VØ]=

((X *REAL* *ARRAY*))

V2=TRANS[E2;5Ø;V1]=

((5Ø *INT* *NUMB*) (X *REAL* *ARRAY*))

V3=TRANS[E9,K;V2]=

((K *INT* *VARI*) (5Ø *INT* *NUMB*) (X *REAL* *ARRAY*))

V4=TRANS[E3;11ØL;V3]=

((11ØL *DO*) (K *INT* *VARI*) (5Ø *INT* *NUMB*) (X *REAL* *ARRAY*))

V5=TRANS[E5;I;V4]=

((11ØL *DO* I) (I *INT* *VARI* *CLOSED*) (K *INT* *VARY*)
(5Ø *INT* *NUMB*) (X *REAL* *ARRAY*))

V6=TRANS [E6;1;V5]=

((11ØL *DO* I) (1 *INT* NUMB ) (I *INT* *VARI* *CLOSED*) (K *INT*
*VARI*) (5Ø *INT* *NUMB*) (X *REAL* *ARRAY*))

V7=TRANS[E6;K;V6]=

((11ØL *DO* I K)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*) (K *INT*
*VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))
V8=TRANS [E15;X;V7]=

((11ØL *DO* I K)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT*
*VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V9=TRANS[E1Ø;I;V8]=

((11ØL *DO* I K)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT*
*VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V1Ø=TRANS[E7;11ØL;V9] =

((11ØL *EXIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT*
*VARI*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V11=TRANS[E13;Y;V1Ø]=

((Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT*
*VARI*) (K *INT* *VARI*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))
```

```
V12=TRANS[E1Ø;Ø;V11]=

((Ø *INT* *NUMB*)(Y *REAL* *VARY*)(11ØL *EXSIST* *CLOSED*)(1 *INT*
*NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(5Ø *INT**NUMB*)(X *REAL*
*ARRAY*))

V13=TRANS[E13;Z;V12]=

((Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST*
*CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT**VARI*)(5Ø *INT*
*NUMB*)(X *REAL* *ARRAY*))

V14=TRANS[E1Ø; Ø;V13]=

((Z *REAL**VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST*
*CLOSED*)(1 *INT**NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(5Ø *INT*
* NUMB*)(X *REAL* *ARRAY*))

V15=TRANS[E3;12ØL;V14]=

((12ØL *DO*)(Z *REAL* *VARI*)(Ø *INT**NUMB *)(Y *REAL* *VARI*)(
11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT*
*VARI*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V16=TRANS[E5;I;V15]=

((12ØL *DO* I) (Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(
11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB *)(I *INT* *VARI*)(K *INT*
*VARI *

V17=TRANS[E6;1;V16]=

((12ØL *DO* I)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(
11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K
*INT* *VARI*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V18=TRANS[E6;K;V17]=

((12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT**NUMB*)(Y *REAL**VARI*)
(11ØL  *EXSIST* *CLOSED*)(1 *INT**NUMB*)(I *INT**VARI* *CLOSED*)(K
*INT* *VARI* *CLOSED*)(5Ø *INT**NUMB*)(X *REAL**ARRAY*))

V19=TRANS[E15;X;V18]=

((12ØL *DO* I K)(Z *REAL* *VARI*)(Ø*INT* *NUMB*)(Y *REAL* *VARI*)
(11ØL *EXSIST**CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI**CLOSED)(K
*INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL **ARRAY*))

V2Ø=TRANS[E1Ø;I;V19]=

((12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT**NUMB*)(Y *REAL* *VARI*)
(11ØL *EXSIST**CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K
*INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))
```

```
V21=TRANS[E4;111L;V2Ø]=

((111L)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL*
*VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*
*CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL*
 ARRAY ))

V22=TRANS[E4;12ØL;V21]=

((111L)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL*
*VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT**NUMB*)(I *INT* *VARI*
*CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT**NUMB*)(X *REAL*
*ARRAY*))

V23=TRANS[E4;112L;V22]=

((112L)(111L)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y
*REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT*
*VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X
*REAL**ARRAY*))

V24=TRANS[E7;111L;V23]=

((112L)(111L*EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT*
*NUMB*)(Y *REAL**VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I
*INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X
* REAL* *ARRAY*))

V25=TRANS[E13;Y;V24]=

((112L)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(O *INT*
*NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)
(I *INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø* INT **NUMB*)
(X*REAL* *ARRAY*))

V26=TRANS[E14;Y;V25]=

((112L)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT*
*NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST**CLOSED*)(1 *INT**NUMB*)(I
*INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT**NUMB*)(X
*REAL* *ARRAY*))

V27=TRANS[E15;X;V26]=

((112L)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT*
*NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST**CLOSED*)(1 *INT**NUMB*)(I
*INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT**NUMB*)(X
*REAL* *ARRAY*))

V28=TRANS[E10;I;V27]=

((112L(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT*
*NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I
*INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X
*REAL* *ARRAY*))
```

V29=TRANS[E4;12ØL;V28]=

(112L)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT**VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT**NUMB*)(X *REAL* *ARRAY*))

V30=TRANS[E7;112L;V29]=

((112L *EXIST*)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* * NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST**CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K * INT* * VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V31=TRANS[E13;Z;V3Ø]=

(112L *EXIST*)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT**VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø * INT* *NUMB*)(X *REAL* *ARRAY*))

V32=TRANS[E14;Z;V31]=

((112L *EXIST*)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT**VARI**CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V33=TRANS[E15;X;V32]=

((112L *EXIST*)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI**CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø *INT *NUMB*)(X *REAL* *ARRAY*))

V34=TRANS[E1Ø;I;V33]=

((112L *EXIST*)(111L *EXIST*)(12ØL *DO* I K)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXOST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI* *CLOSED*)(K *INT* *VARI* *CLOSED*)(5Ø* INT* *NUMB*)(X *REAL* *ARRAY*))

V35=TRANS[E7;12ØL;V34]=

((112L *EXIST* *CLOSED*)(111L *EXIST* *CLOSED*)(12ØL *EXIST* *CLOSED*)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y *REAL* *VARI*)(11ØL *EXIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

V36=TRANS[E12;Y;V35]=

((112L *EXIST* *CLOSED*)(111L *EXIST **CLOSED*)(12ØL *EXIST* *CLOSED*)(Z *REAL* *VARI*)(Ø *INT* *NUMB*)(Y * REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(5Ø *INT* *NUMB*)(X * REAL* *ARRAY*))

V37=TRANS[E12;Z;V36]=

((112L *EXIST* *CLOSED*)(111L *EXIST **CLOSED*)(12ØL * EXIST* *CLOSED*)(Z*REAL* *VARI*)(Ø *INT* * NUMB*)(Y *REAL* *VARI*)(11ØL *EXSIST* *CLOSED*)(1 *INT* *NUMB*)(I *INT* *VARI*)(K *INT* *VARI*)(5Ø *INT* *NUMB*)(X *REAL* *ARRAY*))

Revised separatum to CL&CL Vol.X.

ONE MODEL OF THE HUNGARIAN VERB SYNTHESIS

Mrs M.LUGOSI PAP

1. INTRODUCTION

The aim of the present paper is to give a model of the automatic synthesis of the Hungarian verbs on the basis of the work entitled "Grammatical form system of Hungarian word-stock" [2] and to demonstrate some possible applications of the model. It was my aim to formalize the verbal system in a most suitable and a most precise way and to handle several problems in a uniform method. The formation of the simplest verbal forms has been worked out as a program but I constructed the program /the program-details/ in a way which facilitates to complete it to a whole system /derivation of formal varieties, compound and recursive forms/.

The program is constructed for a System 4-70 machine, in Usercode Language, the particular command-set of which made programming easier /e.g. with one command word-elements of arbitrary length can be compared/.

The following method can be applied for the synthesis of Hungarian nominals in an analogous way; since the paper which served as a base [2] deals with nominal forms too and in a similar manner.

The method is not restricted to the Hungarian grammatical form system. In a language with a developed influctional system /e.g. French, German, Russian/ there is a possibility to construct a suffixal system by arranging the verbs according to the formation of their several verbal forms. And in such a system numbering of the suffixes, recursion, comparing of the suffixal types can be applied in the same way as in Hungarian.

2. IT IS NECESSARY TO CLARIFY SOME IDEAS BEFORE DISCUSSING THE PROBLEM

The notion of the verb stem and that of the suffix must be given, since they differ from the traditional definition. The part of the verb which is invariable during conjugation is called the 'v e r b s t e m' - in case of the machine processing -; the 'termination' is the variable part of the verb - but this is often not equal with the personal suffix connected with tense suffix and modal suffix /see [6]/. I have used an even wider notion of termination in order to give the possibility to store the change of stem and certain stylistic comments concerning the verb automatically with help of the suffix. /See further 3.4.2.4, 3.5.3, 3.5.4./

I understand 't e r m i n a t i o n' /'suffix'/ as an /alphanumeric/ character sequence which contains the suffix with a certain comment and with information concerning the verb stem /generalized idea of suffix or termination/.

The EBCDID code which is used to punch the cards of Usercode programs does not contain the special vowels of Hungarian /ö, ü and the long vowels/. But it is by all means necessary to mark them somehow. If we do not want to mark these vowels with an arbitrary letter or a non-letter character not being used in Hungarian, it is only possible to mark these vowels not with one, but with more characters /letters/. The transcription used in telegraphy cannot be applied here because is would result misunderstandings (e.g. "leegyen" might mean: "leegyen" /'eat messily' in subjunctive mood/ and "légyen" /'let it be'/). So we must choose characters which differ from the letters of the alphabet. A solution for this problem can be found in [4], but the characters used there are not found in the EBCDIC code. Thus I have selected the following solution: the length of the vowel is marked by a colon after the vowel /co-ordinates with the designation structure of APhI/ and the two dots of ö, ü are denoted by quotation-marks. /In the case of the long ő the quotation-mark precedes the length-mark/. /e.g.: Á=a, Ö=o", ő=o":/

The number of characters necessary to denote a verb /suffix, verbal form/ for the computer is called the 'l e n g t h o f t h e v e r b'. I will not necessarily be equal with the length of the verb taken in the usual sense because of the special vowels. E.g. the length of the verb "vöröslik" /'appear red'/ covers 8 characters in the traditional way and 10 characters for the computer.

If one code number /see 3.2/ has more verbal forms, we get 's u f f i x s e r i e s' /'paradigm seris'/ where the different suffixes are written side by side and are separated by commas /or parentheses/. E.g. the imperative form of second person in singular, in the present tense /code number: 42/ has two verbal forms: "várj", "várjál" /'wait'/, so the suffix series is: ~j /~jál/.

All characters of the EBCDIC code have a hexadecimal number. Sorting the hexadecimal numbers in order of size and making the parallel characters in the same order, we get the 'm a c h i n e a l p h a b e t i c o r d e r' of characters. This is not equal to the ordinary alphabetic order because in the second case there is no difference between the short and the long vowels (e.g. the order of the vowels is ó, ő, ö, o ; i.e. in the machine alphabetic order the verb "hólyagzik" /'blister'/ stands before the verb "hokizik" /'play hockey'/, although normally they are in reverse order).

The verbal form constituted from two words spelt aside, is called 'c o m p o u n d   v e r b a l   f o r m', e.g. "ettem volna" /'I should have eaten'/; the verbal form conjugated on from an already constructed verbal form of which the base is a stem from the dictionary, is called a 'r e c u r s i v e   v e r b a l   f o r m', e.g. "ad - adhat - adhattam" /'give' - 'may give' - 'I might give'/.

3. LET US NOW TURN OUR ATTENTION TO THE CONJUGATION SYSTEM OF THE VERB AND TO THE PROGRAM BASED ON IT

3.1 INTRODUCTION

3.1.1 The project called "Grammatical form system of Hungarian word-stock" is elaborated by László Elekfy at the Institute of Linguistics of the Hungarian Academy of Sciences, therefore I will call it EL's system for the sake of brevity. He worked out in details the list of words in the Concise Dictionary of Hungarian [5].

Two variants of the system were finished during the years which differ from each other in details. The simpler system /the so-called 'c o n t r a c t e d   s y s t e m'/ was published in 1972 in the periodical 'Hungarian Language' [1]. It contains only 153 conjugational types and denote only the most important differences. "Among the words which have extremely special terminations, only those are represented in the table of types which in certain points of view are more compatible with the system, especially if they show proper complicacy and are not usually dealt with in the grammatical descriptions" /i.e.: the table does not contain the most part of the special conjugational types[x]/. The numbering of the conjugational types also differs from that one shown in para. 3.3.1 .

The detailed system /i.e. 'the f u l l v a r i a n t'/ will be discussed below. This full variant is to be found in a hand-written version. "It may be called complete within a certain scope" /see [1]/.

3.1.2 We must decide which of the two systems will be transformed into a /machine/ program. We use the detailed system for this purpose because the contracted system takes no notice of lesser differences between the conjugational types and therefore it may produce incorrect or non-existent forms.

The question may arise whether it is worth programming the system in a way to produce all the forms which belong to one code number. For example, if we want to employ the system as a subroutine of a machine translating program from a foreign language into Hungarian, it will be enough to produce one /namely the most frequent/ form. Nevertheless I tried to program the system which includes all the verbal forms because of the possibilities to solve additional problems emerging in the course of programming.

3.2 ON THE VERBAL FORMS INCLUDED IN THE SYSTEM

All the simple verbal forms /with the exception of the imperfect tense/ and all the participles used today and all

---

x Those conjugational types are called 's p e c i a l c o n j u g a t i o n a l t y p e s' which contain only one verb. /Among the 515 conjugational types in the detailed system there are 233 special conjugational types - 102 types ending and 131 types not ending in -ik in the third person singular of the present tense./

the derivations of grammatical character /paradigmatic/ are included in the system; each of them was given a 'f o r m n u m b e r' /'c o d e n u m b e r' , as being called in the program/.

In EL's system more verbal forms are denoted by the same form number if they are always changing in the same way. This simplies the description. But a program would be more complicated by such a numbering system, therefore in the program every verbal form will have its own number. /This is called 'd e t a i l e d n u m b e r i n g s y s t e m'/. Code numbers in the program run from 1 to 63.

The project called "Grammatical form system of Hungarian word-stock" deals with other verbal forms too. But these are already recursive forms. Among the derivations the participles marked 54, 55, 56, 60 and the noún-type marked 61 can be conjugated according to one model of declension; and in the same way, the verb-type marked 62 according to the conjugational type 5a and 5b, the verb-type marked 59 according to the conjugational type 5a8 and 5b2, the infinitive marked 40 according to the declensional type 36D and 36B.

Since the further-declined forms of the infinitive with personal suffix /e.g. "adnom", 'give' in the structure: I ought to give/ occur in verbal structures /e.g. "adnom kellene" - 'I should give'/, derivations of these were included in the system.[x] Code numbers from 65 to 70 were given to these verbal forms.

Remark: These forms, however, are conjugated by recursion, otherwise there would appear too many suffixes in the system.

---

x In the course of the following discussion if it is necessary to make a distinction between EL's system and the machine system transformed according to the above and other points, the latter will be called 'm a c h i n e s y s t e m'.

Table 1: contains the forms of the system with their code number.

Furthermore the detailed numbering system is used. A difference from this is only by the quotations from the original conjugational system.

Table 1

Form number	Code number	Mood	Tense	Number	Person	Type of conjugation	Example	
							Hungarian	English
1	1	declarative	present	singular	1.	subjective	várok	I wait
2	2		present	singular	2.	subjective	vársz	you wait
3	3		present	singular	3.	subjective	vár	he/she waits
4	4		present	plural	1.	subjective	várunk	we wait
5	5		present	plural	2.	subjective	vártok	you wait
6	6		present	plural	3.	subjective	várnak	they wait
7	7		present	singular	1.	objective conjugation relating to object of 2.person	várlak	I wait for you
8	8		present	singular	1.	objective conjugation relating to object of 3.person	várom	I wait for him/her
	9		present	singular	2.	-"-	várod	You wait for him/her
9	10		present	singular	3.	-"-	várja	he/she wait for him/her

Table 1. suit

Form number	Code number	Mood	Tense	Number	Person	Type of conjugation	Example	
							Hungarian	English
10	11	declarative	present	plural	1.	objective conjugation relating to object of 3.person	várjuk	we wait for him/her
11	12		present	plural	2.	-"-	várjátok	you wait for him/her
12	13		present	plural	3.	-"-	várják	they wait for him/her
13	14		past	singular	1	subjective	vártam	I waited
	15		past	singular	2.	subjective	vártál	you waited
14	16		past	singular	3.	subjective	várt	he waited
15	17		past	plural	1.	subjective	vártunk	we waited
	18		past	plural	2.	subjective	vártatok	you waited
16	19		past	plural	3.	subjective	vártak	they waited
17	20		past	singular	1.	objective conjugation relating to object of 2. person	vártalak	I waited for you
	21		past	singular	1.	objective conjugation relating to object of 3.person	vártam	I waited for him/her
	22		past	singular	2.	-"-	vártad	you waited for him/her
18	23		past	singular	3.	-"-	várta	he/she waited for him/her
19	24		past	plural	1.	-"-	vártuk	we waited for him/her
	25		past	plural	2.	-"-	vártátok	you waited for him/her
20	26		past	plural	3.	-"-	várták	they waited for him/her
21	27	conditional	present	singular	1.	subjective	várnék	I'd wait
22	28		present	singular	2.	subjective	várnál	you'd wait
23	29		present	singular	3.	subjective	várna	he/she would/should wait
24	30		present	plural	1.	subjective	várnánk	we would/should wait
	31		present	plural	2.	subjective	várnátok	you'd wait

Table 1. suit

Form number	Code number	Mood	Tense	Number	Person	Type of conjugation	Example: Hungarian	Example: English
25	32	declarative	present	plural	3.	subjective	várnának	they would/should wait
26	33		present	singular	1.	objective conjugation relating to object of 2.person	várnálak	I'd wait for you
	34		present	singular	1.	objective conjugation relating to object of 3.person	várnám	I'd wait for him/her
26	35		present	singular	2.	objective conjugation relating to object of 3.person	várnád	you'd wait for him/her
27	36		present	singular	3.	-"-	várná	he/she would/should wait for him/her
28	37		present	pular	1.	-"-	várnók, várnánk	we would/should for him/her
	38		present	pular	2.	-"-	várnátok	you'd wait for him/her
29	39		present	pular	3.	-"-	várnák	they would/should wait for him/her
30	40	infinitive					várni	to wait
31	41	subjunctive	present	singular	1.	subjective	várjak	/'that I wait'/
32	42		present	singular	2.	subjective	várj, várjál	/'that you wait'/
33	43		present	singular	3.	subjective	várjon	/'that he/she wait'/
34	44		present	plural	1.	subjective	várjunk	/'that we wait'/
	45		present	plural	2.	subjective	várjatok	/'that you wait'/
35	46		present	plural	3.	subjective	várjanak	/'that they wait'/
36	47		present	singular	1.	objective conjugation relating to object of 2.person	várjalak	/'that I wait for you'/
	48		present	singular	1.	objective conjugation relating to object of 3.person	várjam	/'that I wait for him/her'/
37	49		present	singular	2.	-"-	várd, várjad	/'that you wait for him/her'/
38	50		present	singular	3.	-"-	várja	/'that he wait for him/her'/

Table 1. suit

Form number	Code number	Mood	Tense	Number	Person	Type of conjugation	Example: Hungarian	Example: English
39	51	subjonctive	present	plural	1.	objective conjugation relating to object of 3.person	várjuk	/'that we wait'/
	52		present	plural	2.	-"-	várjátok	/'that you wait'/
	53		present	plural	3	-"-	várják	/'that they wait'/
40	54	continuous /present tense,1./ participle					váró	waiting
41	55	perfect /past tense, 2./ participle					várt	waited
42	56	future /future tense,3./ participle					várandó	/to be waited for/
43	57	simultaneous presented mood /1/ adverbial participle					várva	waiting
44	58	antecedent presented cause /2./ adverbial participle					várván	
45	59	Hungarian verb formed with the suffix '-hat' or '-het'					várhat	may wait
46	60	Hungarian participle formed with the suffix '-ható' or '-hető'					várható	may be waited
47	61	verbal noun					várás	waiting as a noun
48	62	causative verb					várat	make sy wait
49	63	passive verb					váratik	/is waited for/
	65	gerund with personal suffix		singular	1.		várnom	for me to wait
	66	gerund with personal suffix		singular	2.		várnod	for you to wait
	67	gerund with personal suffix		singular	3.		várnia	for him/her to wait
	68	gerund with personal suffix		plural	1.		várnunk	for us to wait
	69	gerund with personal suffix		plural	2.		várnotok	for you to wait
	70	gerund with personal suffix		plural	3.		várniuk, várniok	for them to wait

3.3 SYSTEMATIZATION AND CLASSIFICATION OF THE VERBS

3.3.1 In EL's system

According to the conjugational system the verbs are devided into groups of conjugational types, each of them has a 'conjugational - type - number' consisting of 5 alphanumeric characters:

$$a_1 a_2 a_3 a_4 a_5$$

a/ The verbs may be classified into 20 groups, according to the following features:

i/ If the verb does not end in 'ik' in the 3rd person singular of the present tense, $a_1 a_2 \leq 10$; if the verb ending '-ik' in the 3rd person singular of present tense, $a_1 a_2 \geq 11$.
Let us mark now the verbs without 'ik' by: $a_o=0$ and the verb with 'ik' by $a_o=10$.
Remark: the first character of a two-digit number is denoted by a_1, the second by a_2; and the complete two-digit number by $a_1 a_2$ /$a_4 a_5$ will be interpreted similarly/.

ii/ According to the way of joining the suffixes to the stem the following variations are possible /variation is denoted by a'_o/

- the verb is conjugated only by a simple suffix; in this case: $a'_o=1$.
  /e.g. "ír" = 'write' , "múlik" = 'pass'/
- the suffix of the past tense is written to the stem with the help of a vowel; in this case $a'_o=2$.
  /e.g. "tud" 'know' - "tud-o-tt" = 'he knew', "uralkodik" = 'govern' - "uralkod-o-tt" = 'he governed'/

- the suffix of the infinitive and the conditional is written to the stem with the help of a vowel; in this case a'_o=3.
  /e.g. "hall" = 'hear' - "hall-a-nék" = 'I'd hear', "mosdik" = 'wash' - "mosd-a-nék" ='I'd wash'/
- the imperative is not formed with the usual 'j'; in this case a'_o=4.
  /e.g. "olvas" = 'read' - "olvas-s" = 'that you read', "zongorázik" = 'play the piano' - "zongorázz" = 'that you play piano'/
- the imperative is formed from a modification of the lexical stem; in this case a'_o=5.
  /e.g. "fut" = 'run' - "fu-ss" = 'that you run' , "mászik" = 'climb' - "má-ssz" = 'that you climb'/
- the stem ends in an 'l' and there is an elision; in this case a'_o=6.
  /e.g. "gyalogol" = 'go on foot' - "gyalog-lok" = 'I go on foot' , "fuldoklik" = 'choke' - "fuldok-lanak" = 'they choke'/
- there is an elision in the verb and the stem does not end in a 'z' or an 'l'; in this case a'_o=7.
  /e.g. "seper" = "sweep' - "sep-rünk" = 'we sweep' , "ugrik" = 'jump' - "ug-o-rtok" = 'you jump'/
- the verb has an elision and the stem ends in a 'z'; in this case a'_o=8.
  /e.g. "szoroz" = 'multiply' - "szor-z-ott" = 'he multiplied', "hiányzik" = 'be absent' - "hiány-o-ztam" = 'I was absent'/
- the verb is irregular; in this case a'_o=9.
  /e.g. "van" = 'be', "alszik" = 'sleep'/

- the verb is a conjugational type of double conjugation*, without a mixed vowel system** or the verb is very defective; in this case $a_o'=10$. /e.g. "súg-búg" = 'susurrate', "ázik-fázik" = 'be rain-soaked', "gyere" = 'come'/

The type-numbers of the 20 groups may be obtained from the union /sum/ of a_o and a_o'; the group number is the a_1a_2 character in the number of conjugational type.

Remark: If $a_o=10$ and $a_o'=7$, some of the members of these are also defective!

E.g. $a_1a_2=a_o+a_o'=0+8=8$: the group of verbs where the stem ends in 'z' with elision and without 'ik' conjugation.

$a_1a_2=a_o+a_o'=10+8=18$: the group of verbs where the stem ends in 'z', and conjugated with 'ik'.

b/ The verbs may be classified further inside each group, according to the vowel system:

$a_3=a$ if the verb is of a velar vowel system
$a_3=b$ if the verb is of a palatal, unrounded vowel system
$a_3=c$ if the verb is of a palatal, lip-rounded vowel system
$a_3=d$ if the verb is of a double conjugational type of the mixed vowel system.

c/ The classes a,b,c of vowel system, found inside each of the

---

* A verb constructed from two verbs connected with a hyphen is called a 'd o u b l e c o n j u g a t i o n a l' verb, independently of the fact whether the verb is a doublet or it has a co-ordinate structure. Both verbs in a double conjugation, are conjugated seperately and also after conjugation they are connected with a hyphen.

** The double conjugational verb in which the 2 verbs belong to different classes of the vowel system, is a verb of '<u>m i x e d v o w e l s y s t e m</u>'.

20 groups; the class d occurs only in some of the groups. Within the main types /altogether 66/ obtained in this way we can differenciate several subtypes /maximum 15/. The number of subtypes within a main type is denoted by a_4a_5 characters. This number of the subtypes shows how many sub-groups have to be differentiated within the main type in order to give a correct description of the forms of each verb, belonging to the main type. At the end we obtain altogether 515 conjugational types.

Remark: It results clearly what is said above that there are no exceptions in this conjugational system. I stress this fact because like this, the system is more homogeneous and well arranged /and therefore it may be better programmed /see [6]/.

3.3.2 In the machine system

In the machine system the conjugational type-number consists of 6 characters: a_6 contains the systematic remarks concerning the verb /see 3.5.4.2/.

The system is broadened by an URES /='EMPTY'/ conjugational type in order to describe the very defective conjugational type in a simple way: /see 3.5.1.2/. This is a fictive conjugational type, a verb belonging here to has no single verbal form.

3.4 DESCRIPTION OF THE VERBAL FORMS

3.4.1 In EL's system

3.4.1.1 Marking the suffixes

With the verb type it is after the verbal form that stands the suffix. Signs used at the description of the suffix are the following:

- The suffix after ~ means that the suffix is written to an unmodified lexical stem.
  E.g. "ápol"='cure' belongs to the la type, the suffix of the form 30 in this type is '~nánk'; so the whole verbal form is: "ápolnánk" /='we would cure'/.
- If the suffix is connected to the stem in such a way that the stem is changed then the last unvariable letter of the stem together with the suffix is put after 2 dots /../.
  E.g. "avat" /='dedicate'/ belongs to the conjugational type 5a, the suffix of code number 41 is ..assak, so the verbal form 41 is "avassak".

3.4.1.2 Missing verbal forms

Not each verb has all the 63 verbal forms. The missing verbal forms must be denoted too. A horizontal line after the code number of the verbal form indicates its absence. The absences in the systematic remark /see 3.5.4.1/ are not denoted.

3.4.1.3 Usage of particular verbal forms

It may happen that a code number has more verbal forms as form variants. E.g. the imperative form of the second person singular int the present tense /"várd", "várjad" - 'let you wait for him/her'/ or in the conditional type 3a the conditional forms may be conjugated with a linking vowel or without it /e.g. "körülrajong-a-nék", "körülrajong-nék" - 'I'd admire'/.

The system contains all the possible verbal forms[x] and gives an information automatically for the usage of forms: the suffix included before is more frequent and the suffix following it in the description may be less frequent[xx].

E.g. The type 3a, code number 27: ~anék, vagy ~nék.

The "infrequency" of a form is denoted in such a way that the suffix is in parantheses independently of whether it has a more frequent variant or not.

E.g. the type 3b, code number 62: /~tet/
the type 3al, code number 14: /~tam,~ottam/

However the participle marked 58 which is rare today for all the verbs and the passive voice marked 63 which is very rare and archaistic in the up-to-date standard language are not put into parantheses.

Other stylistic remarks are to be found in 3.5.4 .

---

x But the forms which are very unusual or are to be avoided in the everyday language are not indicated.

xx Here we must interpret the word 'l e s s f r e q u e n t' in a wider sense: it may have the meaning that the form is less desirable, a little rustic, archaic or high-brow differing from usual everyday language but is not ungrammatical.

3.4.2 In the machine system

3.4.2.1 The suffixes connected to the stem

In the course of programming the suffixes connected to the stem do not give rise to difficulties: the information corresponding to the suffix is attached directly to the stem.

3.4.2.2 The suffixes not directly connected to the stem

If during the conjugation the stem is changed, the solution described in 3.4.1.1 /with the 2 dots/ is unsuitable because it is based on the linguistic instinct of the native speaker and such a linguistic instinct is not to be expected from a machine /cf.[7]/. Thus the change of stem must be marked formally.

The system of forms conjugated with a changed stem may be divided into two groups:

1. the difference may be specified according to some system /e.g. elision/,
2. there is no a system like this /e.g. irregular verbs/.

To except the program for examination of how the suffixes are connected to the stem, would not be saving; therefore the change of the stem is denoted by the first character of the generalized suffix. Since we must differenciate between the generalized suffix and the real one, the first character of the generalized suffix may not be such a letter that may occur as the first letter of a real suffix.

The change of the stem often manifests itself as a growing shorter of the stem. It is from this fact that derives the inspiration that the first character of the generalized suffix be the number of characters with which the changed stem becomes shorter. From the second character the real suffix is to be found /not taking into account occasional remarks, see 3.5.4/.

E.g. This means in the case 3.4.1.1, that the suffix of "avat" of the form marked 41 is lssak, thus "avat+lssak" → "ava+ssak= avassak".

3.4.2.3 Marking of elision

a/ In the case of verbs without 'ik' the notion of elision means that the last vowel of stem is not present in the conjugated form, e.g.

(i) "csicsereg" /'he chirps'/, but "csicsergek" /and not: "csicseregek"/ /I'chirp'/.

(ii) "kevesell" /'he finds sg. too little'/ but "keveslem" /'I find sg.too little'/

If we apply the solution described in 3.4.2.1, we must include a proper suffix to each possible final consonant of the stems.

E.g. csicsereg + 2gek → csicsergek
kicsinyel + 2lek → kicsinylek

Thus we find about 350 suffixes and 280 suffix series.

But it is characteristic of all the different final consonants of the stems that it is set in place of the last but one letter /=the vowel/ and after it comes the suffix. Thus these may be elaborated on the basis of the same principle: Let the first character of generalized suffix be 'X', this indicates that the last letter must be put in place of the last but one and the characters after 'X' denote the real suffix /not taking into account occasional remarks, see 3.5.4/.

E.g. csicsereg + Xek → csicserg + ek = csicsergek
kicsinyel + Xek → kicsinyl + ek = kicsinylek

In this way only about 160 suffixes are necessary to the description of these forms.

The elision of type (ii) can be found only in two conjugational types /7bl, 7b6/, so it is not worth to assign a separate letter for them and to write a separate program because it would not decrease the number of suffixes - therefore this types are handled in the way described in 3.4.2.2.

b/ In the case of verbs with 'ik' the notion of elision means that a vowel is inserted in the stem inside the lexical form.

E.g. "romlik" /'spoil'/ but "romoltok" /'you spoil'/. It is characteristic for this type of elision that a vowel corresponding to the vowel system is interpolated between the last letter of the verb and the last but one: in the case verbs containing velar vowels: "o", in the case of verbs with palatal unrounding vowel: "e" and in the case of verbs with palatal lip-rounded vowel: "ö".

Let "Y" be the first character of the generalized suffix, it marks the epentheses and the characters following "Y" will mark the real suffix.

E.g. "ugrik" /'jump'/, /17a/, code number 5:
ug|rik + Ytok → ugor + tok = ugortok
"vérzik" /'bleed'/, /18b/, code number 6:
vérz|ik + Ynek → vérez + nek = véreznek
"ömlik" /'flow'/, /16c/, code number 59:
öml|ik + Yhet → ömöl + het = ömölhet .

This solution has the advantage of shortening all the suffixes by 2 or 3 characters.

3.4.2.4 Shortening and lengthening of vowels

The shortening and lengthening of vowels may be treated simply and similarly on the basis of the designation of "special" vowel, the principle described in 3.4.2.2. Namely the shortening of vowels may be considered such a change of the root where the length of verb decreases with 1 /viz. with the character ':' which denotes the length of the vowel/. The uniform way of dealing with the problem means that the shortening may be denoted independently from the shortened vowel /this is impossible in the case of the usual designation of vowel-length in Hungarian/.

E.g. "nyű" /'wess out'/ but "nyüvés" /'wessing out'/;
"sző" /'weave'/ but "szövés" /'weaving'/,

but we obtain both forms with the same suffix:

nyu": +1ve:s ⟶ nyu" + ve:s = nyüvés
szo": +1ve:s ⟶ szo" + ve:s = szövés

The lengthening of vowels is denoted by a suffix which has a colon as first character.

E.g. "lesz" /'will be'/, code number 58:
lesz + 2:ve:n ⟶ le:ve:n = lévén

3.4.2.5 For the designation of the infrequency of verbs see para 3.6.4.

3.5 THE DESCRIPTION AND PROGRAMMING OF THE CONJUGATIONAL TYPES

3.5.1.1 In EL's system

It would be redundant to describe all the forms in each conjugational type, because not all the forms differ from each other. Conjugational types 1a and 1b are considered as 's t a n d a r d t y p e s'; all the other types are described in EL's system as compared to these.
Namely: in general, in the case of a_3=a we refer to the type 1a, in the same way in the case of a_3=b,c to the type 1b, in the case of a_3=d to the type 1a, 1b.

In such a case we list only those forms which differ from the forms of the types to which they are referred.

E.g. [4a botoz] As 1a, but: 2~ol; 9-12:~za etc.;
14,41 ~ott ! 31-39~zak etc.
/"botoz"='flog'/

3.5.1.2 In the machine system

In the original description the conjugational system is unfitted for programming: it gives exactly the conjugational types but there is no possibility to sum up the types easily and formally. Moreover in the case of verbal forms which change in the same way it gives only the first forms. Therefore I have put the system into a tabular form /see I.Appendix/.

In the machine system each conjugational type has a record of a length of 64 bits /these records are placed into a disk file called DISELT/, to each code number a bit is accorded with an appropriate ordinal number. The value of the bit is 1 or 0 according to whether the suffix differs from the respective code number of an other conjugational type or not. The value of the remaining 64th bit determines whether the type was related to a standard type / in this case the value of the bit is 0/ or to another type /in this case the value of the bit is 1/. A suffix number was given to all the possible suffixes /suffix series/.

Each conjugational type has an other record with a maximal length of 194 bytes /these records are also placed into a disk file called DISMIN/. This record contains the different forms /without the code numbers/, in increasing order according to the code numbers; and the conjugational type too, if the conjugational type is not related to a standard type.

The algorithm of the search of the suffix number is following: We examine the value of the bit corresponding to the code number in the record which belongs to the conjugational type of verb on the DISELT file. If the value is 0, we must examine to which conjugational type it was related and then we examine the value of the bit of this conjugational type, etc.

If the value of the bit is 1, we must count the bits, the value of which is one from the first bit upto this one and the suffix number of the required form will be a suffix number on the DISMIN file, the serial number of which agrees with the number we have obtained.

E.g. Let us look for the form 55 of the verb "tud" /'know'/ which belongs to the type 2a6.
The record related to this type on the DISELT file is:
2a6: 00...............0...............0111
and on the DISMIN file:
2a6: 021 240 2a
The value of the 55th bit in the record of DISELT file is 0, the value of the 64th bit is 1. The 64th bit is the third bit in this record that is equal to 1, and the third data in the record of DISMIN file is: 2a.

Hence we must examine data of the type 2a.
The record in the DISELT file related to this type is:

```
                  16          55
2a: O..........OlO..........OlO..........O
```

Here the value of the 55^{th} bit is 1 and this is the second bit that is equal to 1, therefore we examine to the second suffix number of the record 2a on the DISMIN file and this is: 120. /The suffix number 120 corresponds to the suffix "~ott", thus the form 55 of the verb "tud" is "tudott" /'known'/.

3.5.2 Recursion

3.5.2.1 In EL's system

a/ It may happen that two conjugational types following each other, differ in only some verbal forms but they differ very much from the standard type: in this case we refer to the group which has the smaller type number. This is called '/simple/ r e c u r s i o n'.

E.g. [lcl dől] As lc, but intransitive: 42/~endő/! 46,48:-
/"dől"='fall'/

b/ There is no reference to a former or standard type in the case where the subtype is too defective: then only the existing forms are described.

E.g. [10a3 szabad] 3~, 14/~ott/, 23~na, 33~jon
/"szabad"='allowed'/

In the machine system this type may be described only as related to the standard type, e.g. on the DISELT file:

```
                    16      29          43        63
10a3: 11011.......1.......101...........101.........10
```

and the length of the record 10b2 would be 180 bytes on the DISMIN file.

In order to the conjugational type, 'URES' was initiated for simpler administration, thus the description of type 10a3 is: on the DISELT file:

16 29 43
001000 .....010.........010.........010.......01;

on the DISMIN file:

001, 620, 035, 104, URES

3.5.2.2 In the machine system

Programming provides an opportunity to include not only simple but repeated recursion. A repeated recursion is not adequate to "the human utilization" e.g. in a conjugational dictionnary because this indicates too much searching about. On the other hand it gives no problem for the computer. Thus the particular conjugational types were described related to the conjugational types "nearest to" them. /"Nearest to" means, that the distance of two conjugational types is the smallest; "the distance of two conjugational types" is defined as the number of verbal forms differing from each other./ E.g. the type 3c2 may be described related to the type 3c1, the type 3c1 to the type 3b8 and the type 3b8 to the type 1b: this is a 'd o u b l e r e c u r s i o n'.

To 70% of the conjugational types, simple to six times recursion may be applied and so the number of differences can be decreased to half; this means that we need half space on the DISMIN file.

3.5.3 References

3.5.3.1 In EL's system

It may occur in case of some conjugational types that formal variants are used instead of certain verbal forms /references of "p r e f e r a b l e t y p e" or the verb has only some forms and the other forms are expressed by the corresponding forms of a formal variant /of the same verb/ /references of "i n s t e a d o f t y p e" ✱.

See I.Appendix [12a4 mosakodik] and [19a furakszik]
/"mosakodik"='wash',"furakszik"='push'/

3.5.3.2 In the machine system

Considering for small number of "preferably type" references they are transformed according to 3.4.2.2 and in the same way the "instead of type" references in the main type 17.

E.g. the forms 1-4 of the verb "mosakodik" /12a4/ in the machine system:

1..kszom/~om/, 2..kszol/~ol v.~sz/, 3..kszik/~ik/,
4..kszunk v.~unk

In the case of "instead of type" references in the main type 19 this solution would give about 300 further suffixes, therefore I selected an other solution.

For the forms which are conjugated from changed stems, the first character of the generalized suffix is P and the next characters depend on the change of the stem and on the number of the new conjugational type. /There is no real suffix beginning with the character P./ To be more exact: the second character of the suffix denotes the number of the characters to be cut off the end of the verb; and the following characters must be set after the stem derived in such a way. /If these are less then 4, the other characters are replaced each with a space./ The 7th character of the suffix denotes the conjugational type of the stem variant. Hence, we need only 11 new suffixes.

✱ In the following only these will be called "r e f e r e n c e s"

3.5.4 Remarks

3.5.4.1 In EL's system

In order to give an exact and simple description of Hungarian verbs certain remarks are by all means necessary which may be divided into three large groups:

- Remarks type I: these are so-called "s y s t e m a t i c a l r e m a r k s" concern well-defined forms of certain conjugational types. They are denoted in the course of the description of the conjugational system.
  E.g. "intransitive"='tn' and a corresponding conjugational type: e.g.: 2a8.
  The systematical remarks apply to all the verbs belonging to the given conjugational type, as in the former case: e.g. "fagy" /'it freezes'/, "fogy" /'grow less'/.

- Remarks type II: These are similar remarks as those in remarks type I plus the remark: "without subject", but these are not concerning the conjugational type but only certain verbs. These are denoted only in the dictionnary.
  E.g. "lapul intransitive" /'become flat'/. This verb belongs to the type la./
  /These remarks are also called systematical remarks./
  The verbs with the remark "only intransitive" lack the forms 7-13,20-26,33-39,47-53,63 and the forms 56,60 are rare with them.
  The verbs with the remark "only in 3rd person" have only the following forms: 3,6,10,13,16,19,23,26,29,32,36,39,43, 46,50, 53-63.
  The verbs with the remark "only transitive" have no forms 1-6, 14-19, 27-29, 41-46, 55,60,63.

- Remarks type III: These concern certain forms of certain verbs and they are to be found in the foot notes. They may be divided into two large groups:

a/ so-called "c o m m o n r e m a r k s" which concern all the forms with the code number "certain" of certain conjugational types.
E.g. "without 'ik' specially in transitive usage".

b/ so-called "s p e c i a l r e m a r k s" which concern only certain forms of certain verbs in a certain conjugational type.
E.g. 3 / ik/ in the verbs "retten" /'recoil'/,
"rezzen" /'rustle'/, "csökken" /'decrease'/.
/This special remark concerns the conjugational type lb./

3.5.4.2 In the machine system

- The remarks type I, II are contained by the 6^{th} character of the conjugational type number. /If there is not such a remark for a conjugational type, a_6 is a "space" character./
- The remarks type III. are denoted in the generalized suffix by a special character after the last letter of the real suffix which may be well seperated from the last letter of all the real suffixes. The program must direct that only the real suffix be connected to the stem. Of course, the suffixes with a remarks have other suffix numbers than the suffixes without remarks.

3.6 NUMBERING AND STORING OF THE SUFFIXES

3.6.1 General principle

On the base of all the /generalized/ suffixes and suffix series that are possible in the conjugational system, a suffix table was made. Its details may be found in Table 2.

The firs part of the suffix table contains the rare and the frequent suffixes in the machine alphabetic order in the way that the longer suffixes have greater suffix numbers - i.e. the length of the suffix may be determined depending on the value of the suffix number - and in this way the program will be simpler.

In order to occupy less place in counting the suffixes, the suffix number contains only 3 characters in spite of the fact that there exist about 3000 suffixes /and suffix series/. The second and third character of the suffix number go from 00 to 99, the first /alphanumeric/ character of the suffix gives the value of the hundreds.

The second part of the suffix table contains the suffix numbers with respect to the suffix series together with the suffix number of the forms in the above discussed order. A suffix series has 2, 3 or 4 suffix numbers depending on the number of the verbal forms, they are the suffix numbers of these forms.

3.6.2 Storing of the suffixes

The suffixes and suffix series without their suffix number may be found on the DISRAG file in an increasing order. /This file is also an indexed sequential file on the disk./

Table 2

DETAIL OF THE FIRST PART OF THE SUFFIX TABLE

Machine description of the suffix	Remark	Length of the suffix	Frequent suffix			Non-frequent suffix		
			Type in which it can be found first	Suffix number	Code number	Type in which it can be found first	Suffix number	Code number
space	-	1	1a	001	3	7b6	F01	3
space 8	yes	2	1a1	014	3			
space 9	yes	2	1b5	015	3			
:K	-	2	5a7	013	13			
:TOK	-	4	5a7	235	12			
Ø	-	1	1a1	000	60			
ØØ	yes	2	3a1	016	56			
ØH	yes	2	13a	017	62			
ØP	yes	2	19a1	018	60			
			19a1		62			
A:L	-	3		075		14a6	F75	42
A:N	-	3	9a8	076	58			
A:S	-	3	1a	077	61			

Examples for suffixes of Table 2:

ápol +u= ápol /'he/she nursey'/
meggyón +u= meggyón, meggyón + ik = meggyónik /'he confesses'/ in the transitive usage rather without '-ik'
aszongya + :K = aszongyák /'they say'/
aszongya + :TOK = aszongyátok /'you say'/
elbúsul + Ø - the verb "elbúsul" /'he abandons himself to sorrow'/ has no the form 60
the verbs with the suffix number ØØ: the verbs belonging to the type 3a1 have no forms 56, except for the verb "mond" /'say'/: "mondandó" /'saying'/
szív + A:N = szíván /'smoking'/
ápol + A:S = ápolás /'nursing'/

3.6.3 Advantage of the numbering of the suffixes

1/ It claims less place in the data storage. Namely there are about 5600 differences in the conjugational system; in order to mark the differences 5600 x 3 ≈16K bytes are necessary in the case of counting of the suffixes; the data storage of the DISRAG file is about 15K; this is altogether about 31K bytes.

But if we do not count the suffixes, since the average length of the suffixes is 5,4 bytes, in order to mark the difference it is necessary to have 5,4 x 5600 ≈ 30K bytes. /The real data storage, however, needs more place because we have counted suffixes instead of suffix series./ But to find a given suffix it is necessary to give each suffix a seperated data length and thus data storage is more than 50K bytes.

2/ File-handling is simpler because each data is 3 bytes long in each record of the DISMIN file.

3/ If we do not mark the suffixes with a number, it will be necessary to examine whether a code number has one or more forms; and in the former case this follows automatically.

4/ If we want to write a program that makes more or less /i.e. not all/ forms, then we must rewrite only the suffix numbers on the DISMIN file - while if we do not number the suffixes, it would be rewriting the complete file.

5/ Also from the point of making linguistic statistics or an occasional analysing program it is better to number the suffixes. E.g. in the former case it is simpler to examine items with the same length /e.g. to count the frequency of certain suffixes/.

3.6.4 The suffixes of the "rare" forms

The "rare" forms are denoted also by the suffix. If we want to mark the rarity of the forms with a character of the suffix we would get about twice as much suffixes and it would not be economical. Therefore the "rare" suffixes were also given a suffix number which differs from the suffix number of the suffixes of the frequent form, but the suffix number of the frequent suffix can be decided from the first character of the suffix number of the rare form.

3.7 DICTIONARY OF THE CONJUGATIONAL SYSTEM

3.7.1 Dictionary that belongs to EL's system

All the verbs taken into account belong to one type of the 515 conjugational types. However it can not always be determined formally, to which one. Since it is necessary to make a dictionary which contains the verbs in their lexical forms /present tense, 3rd person singular/, together with their conjugational types and accidental remarks. This dictionary is necessary for the usage of the conjugational system.

The verbs may be divided into three groups:

I. The verbs which may be conjugated according to their second part and the second part is a lexical entry - it is not necessary to indicate the conjugational type of these verbs in the dictionary.

II. The verbs with a typical termination which obtain the paradigmes in accordance with the termination. These verbs are called 'Verbs with a typical termination'. A so-called 'Table of typical terminations' belongs to the conjugational system. /See II.Appendix./ It contains the the typical terminations together with their conjugational types.

III. All the other verbs are so-called 'verbs with their own paradigm'. Their conjugational type must be given by all means in the dictionary.

3.7.2 The dictionary belonging to the machine system

1/ The dictionary made according to the machine synthesis contains only the verbs which must be included in order to find their conjugational type, i.e. it does not give a full verb-listing. This dictionary contains only the verbs of the IIIrd group as well as the verbs with prefix belonging to the base verb with the remark "only intransitive".

It is possible for a verb to represent more /homonymus/ lexemes and the different lexemes belong to several conjugational types.

E.g. the verb "kiötlik^{1}" /familiar or coloquial - 'stick out'/ belongs to the type 16c4 and the verb "kiötlik^{2}" /'think out'/ belongs to the type 16c2.

In this case the verb must be included in both conjugational types in a way to be differenciated, since the key of a record of the DISTAR file is the verb itself, such verbs have two keys, e.g. "kiötlik1" and "kiötlik2".

2/ The dictionary is on the DISTAR file which is an indexed sequential file on the disk and the key of the verbs is the verb itself.

3.7.3 Number of verbs in the dictionary

The conjugational dictionary contains about 16000 verbs, half of them belonging to group I. According to the contracted system there are about 6100 verbs with the characteristic termination belonging to group II, thus the type of only 1900 verbs must be given.

However, several types of the detailed system may belong to certain types of the contracted system /e.g. types la, lal,... ...,la9 of the detailed system belong to the type [l] of the contracted system/, but among the frequent terminations only one can be regarded as characteristic, the one that contains the greates number of verbs, thus to a characteristic termination less verbs belong in the detailed system. Therefore the DISRAG file contains not 1900, but about 3600 verbs.

3.8 DOUBLE CONJUGATIONAL VERBS

1/ There are 45 double conjugational types containing 60 verbs.

2/ Both verbs of a double conjugational type have a separated record in the DISELT file as well as in the DISMIN file. The 64^{th} bit of the former record is always 1, because it may not often be decided formally /or would be complicated/ to which standard type it was related by the last number of the recursional change.

The key belonging to the first verb of the double conjugational type on the DISELT file and on the DISMIN file corresponds to the number of the conjugational type and the key belong to the second verb is equal to the key belonging to the first verb, except for the a_3 character; e.g. in the case a_3=d, the new a_3 is "f".

If a form is rare it is denoted in both verbs. The special remark is indicated in the first verb and the common remark is indicated in the second one. If any of the verbal forms are lacking it is denoted only in the record, belonging to the first verb.

3.9 THE DESCRIPTION OF THE PROGRAM

3.9.1 Input data

The program was made in a way that by a relatively simple extension it may formulate the compounds forms /conditional, future tense/. So any forms may be required /i.e. declarative, conditional and imperative, all 3 tenses; the verb ending in "-hat" - e.g. "talál-hat"="találhat"- 'he may find' -, causative forms, reflexive forms/.

It is the task of the program to decide whether the required form is existing at all in Hungarian and if it does, whether it exists for the given verb.

The verb to be conjugated and the required form may be read in from a punch card. On the card the following must be punched:

the verb: by characters 1-35;

the required form: by characters 37-52; this may be given as a code number /its value may be 1-63 or 65-70; in this case only a simple form and an infinitive with a personal suffix may be required/ or a code formed of 16 characters:

$a_1 a_2 . a_4 a_5 a_6 a_7 a_8 a_9 . a_{11} . a_{13} . a_{15} a_{16}$ where

$a_1 a_2$ may be: cs = active voice
mu = causative voice
sz = passive voice
ha = the verb with "-hat"
two spaces
os = all the forms of the verb are demanded

and $a_4 - a_{16}$ contain all the informations concerning the type of conjugation, number, person, participle and other forms gained from the verbs.

3.9.2 Result

The program gives the required form /or the answer that it does not exist/ on the line printer. The first 55 characters of the result are the input data, the other characters depend on the result.

a/ If we have demanded an intransitive verb or a transitive verbal form relating to an object of 2^{nd} or 3^{nd} person, in the case when the required form of the verb exists, the program prints: "A kért igealak:............"
/'The required form: ....'/
/If there exist several forms they are printed one under the other./
If the verb is a rare one, it prints "ritka" /'rare'/ at the end of the line: if there exists any common remark it is printed in the next line.

b/ If we have required a transitive verbal form and we have not specified the type of the object /2^{nd} or 3^{rd} person/ then the program decides whether both forms exist or not:

1. If the verb is intransitive, the printed text is: "Az igének ilyen alakja nincs" /'the verb has not this form'/

2. If the transitive form relating to an object of 2^{nd} person does not exist, it conjugates only the form relating to an object of 3^{rd} person and the printed text is:
"3.személyü tárgyra utaló alak: ...."
/'the form relating to an object of 3^{rd} person'/.
3. If both transitive forms exist, the program will conjugate the form relating to an object of 2^{nd} person in the same way as described in para. 2 and the form relating to an object of 3^{rd} person.

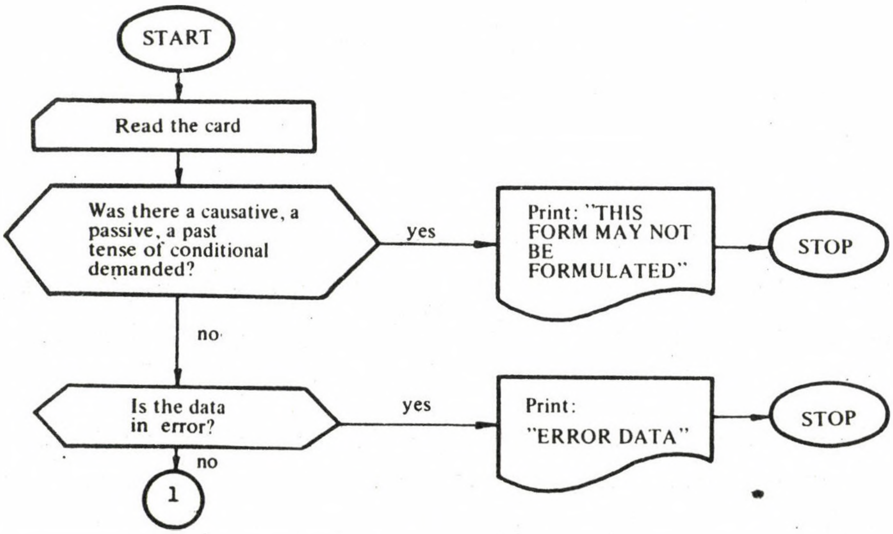
c/ If we require a compound or a recursive form, then the program indicates the understanding of the task but does not conjugate this form yet.

3.9.3 Storage of the program

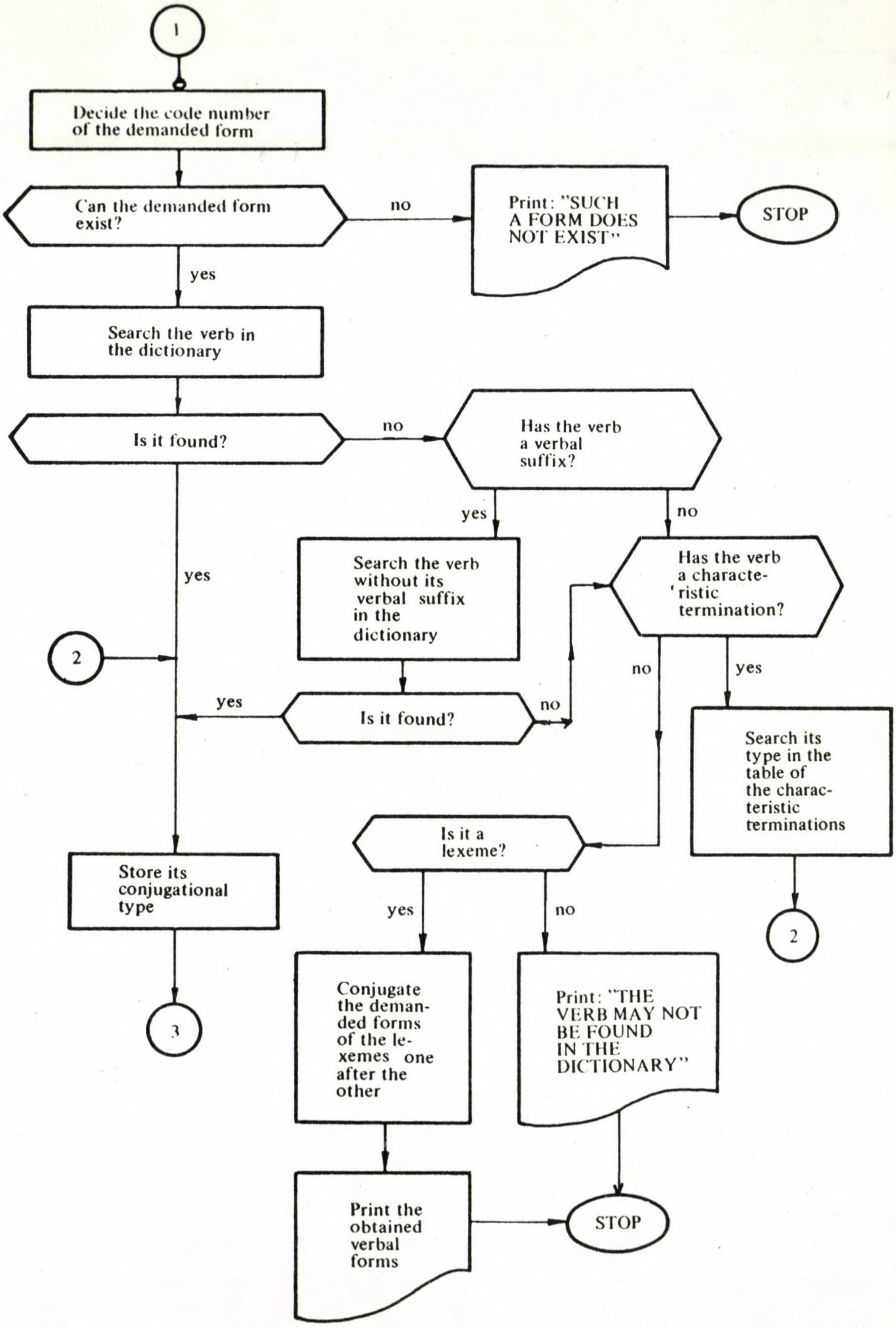
1/ The storage of the DISELT file is about 7K bytes, that of the DISMIN file is 27K *, that of the DISRAG file is 17K *, and that of the DISTAR file is 115K /counting the key area with 25 characters/. This is altogether 166K bytes. If the full DISTAR file becomes ready with about 3600 verbs and the number of characters that define the verb unambiguously, can be decided, it might happen that the key length of a verb covers e.g. only 10 characters. In that way only 63K bytes would be necessary for the DISTAR file.

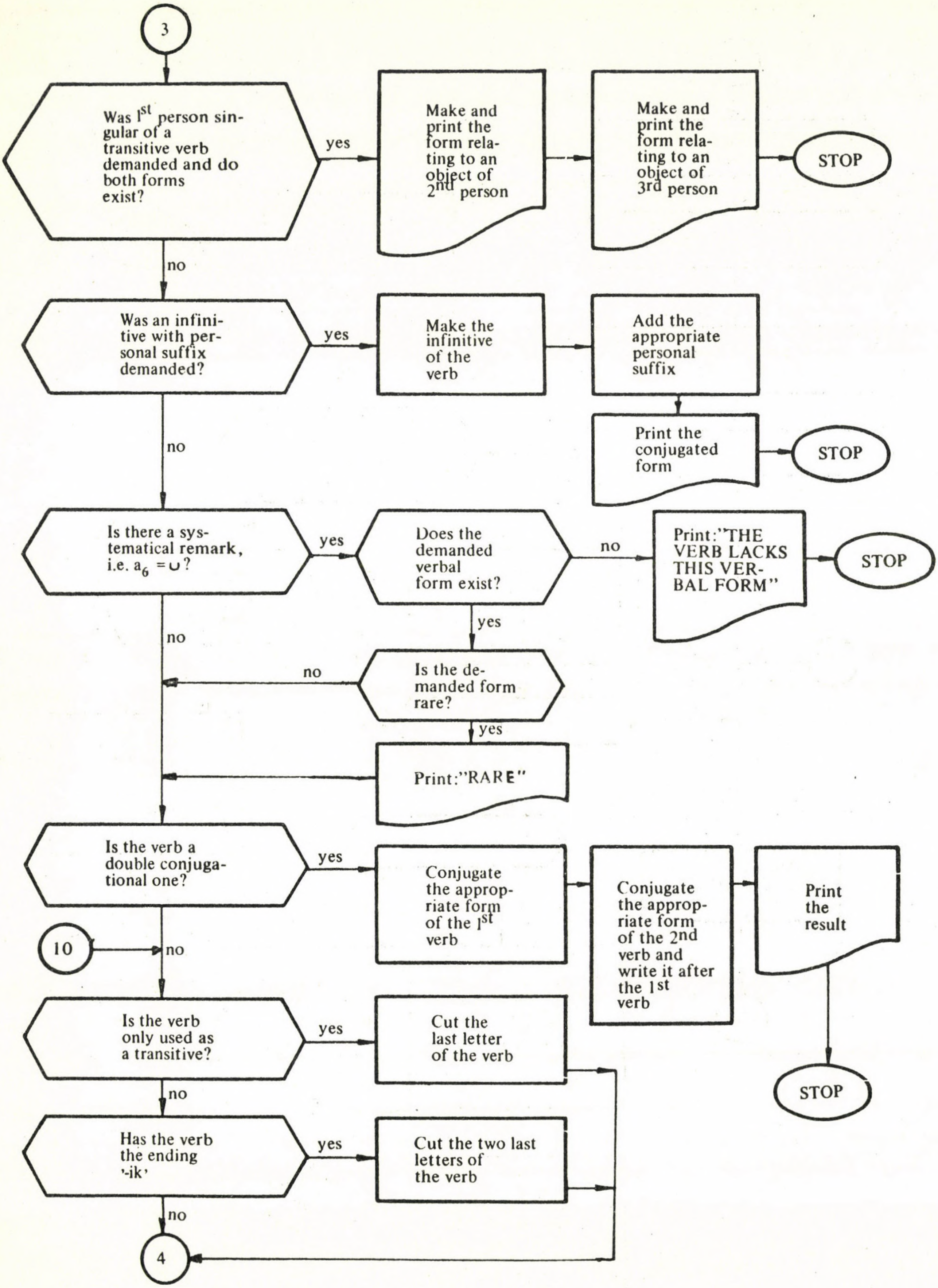
2/ The storage required for the program conjugating the recursive and the compound forms is about 20K.

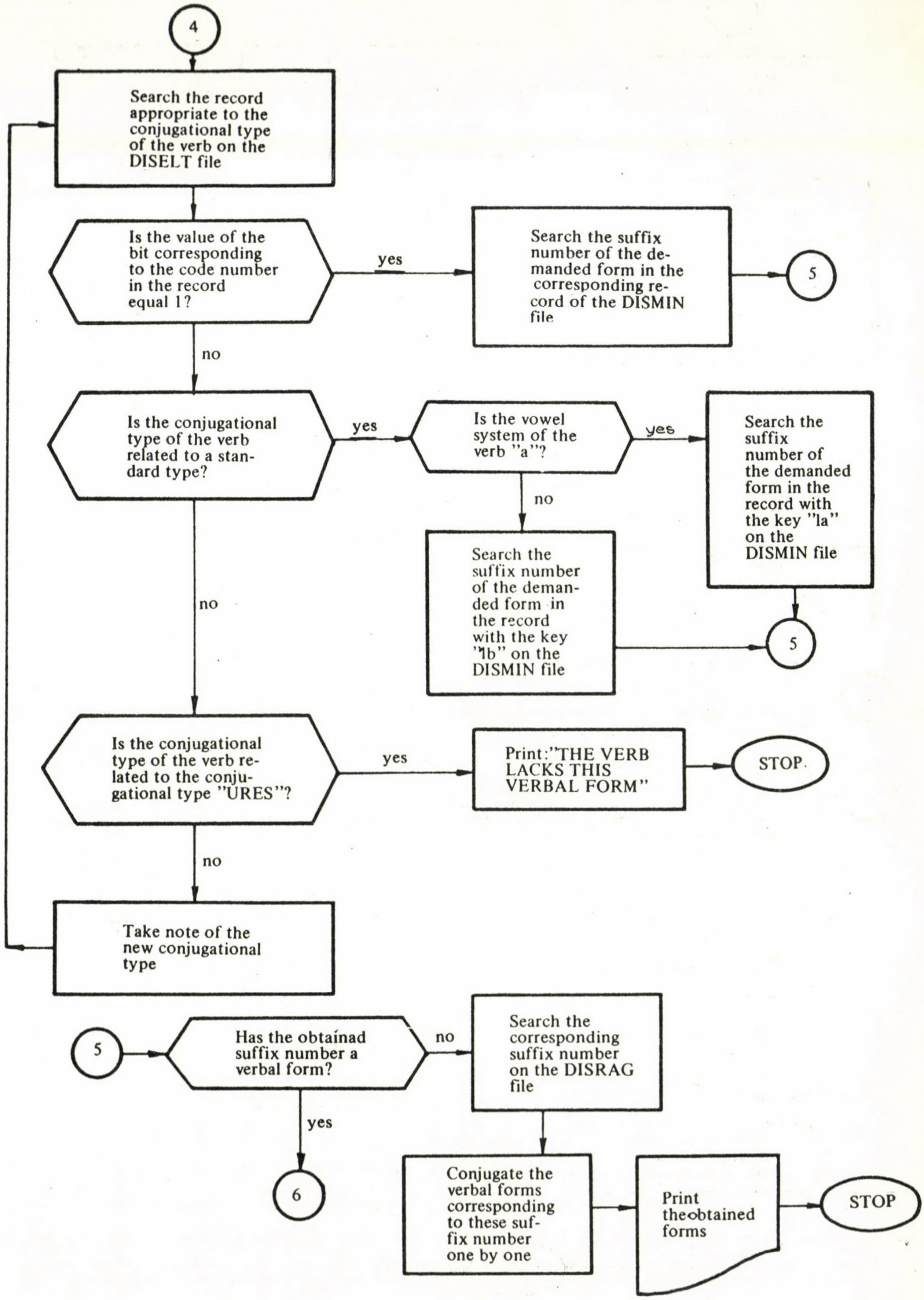
3.9.4 Flow-chart of the program

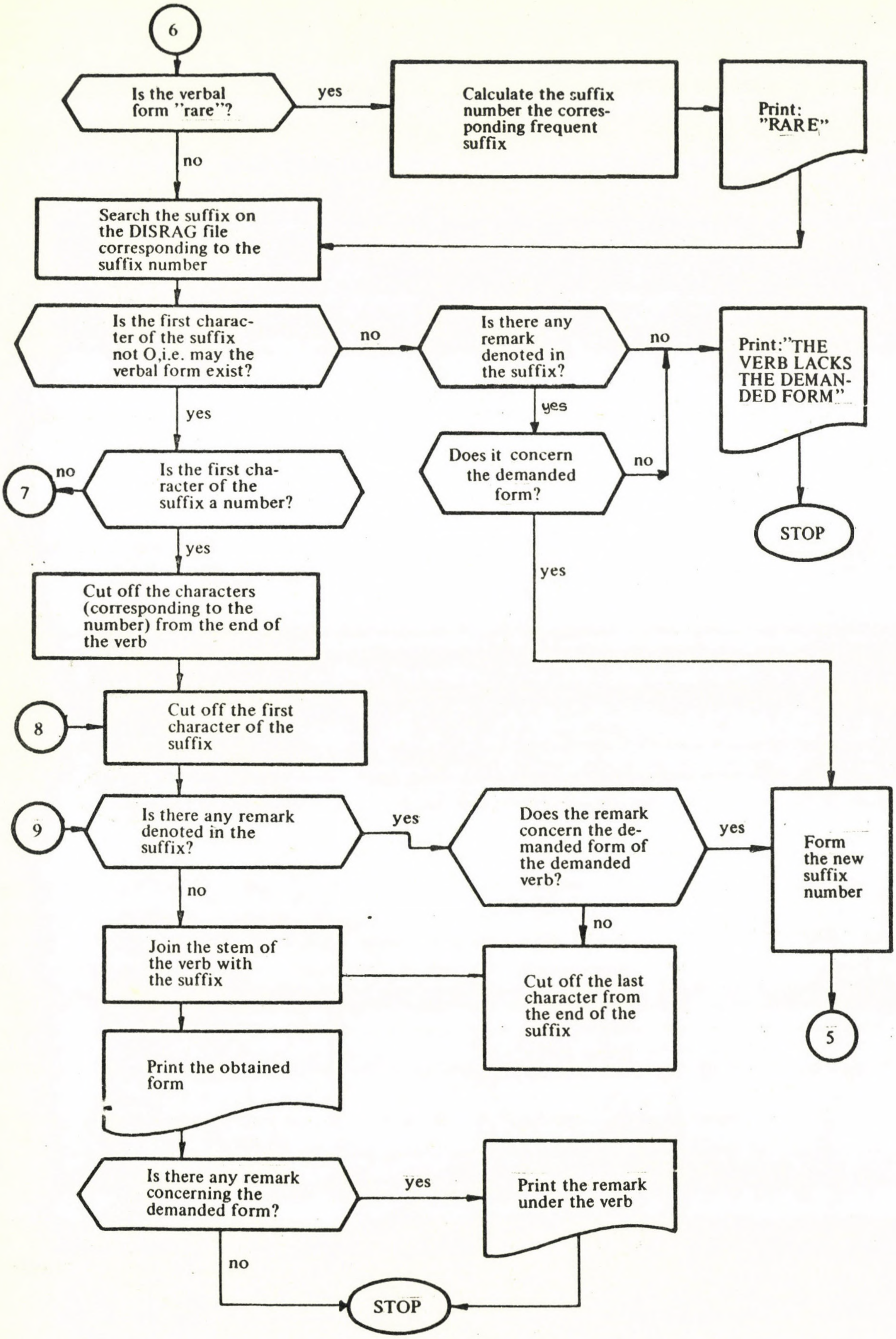


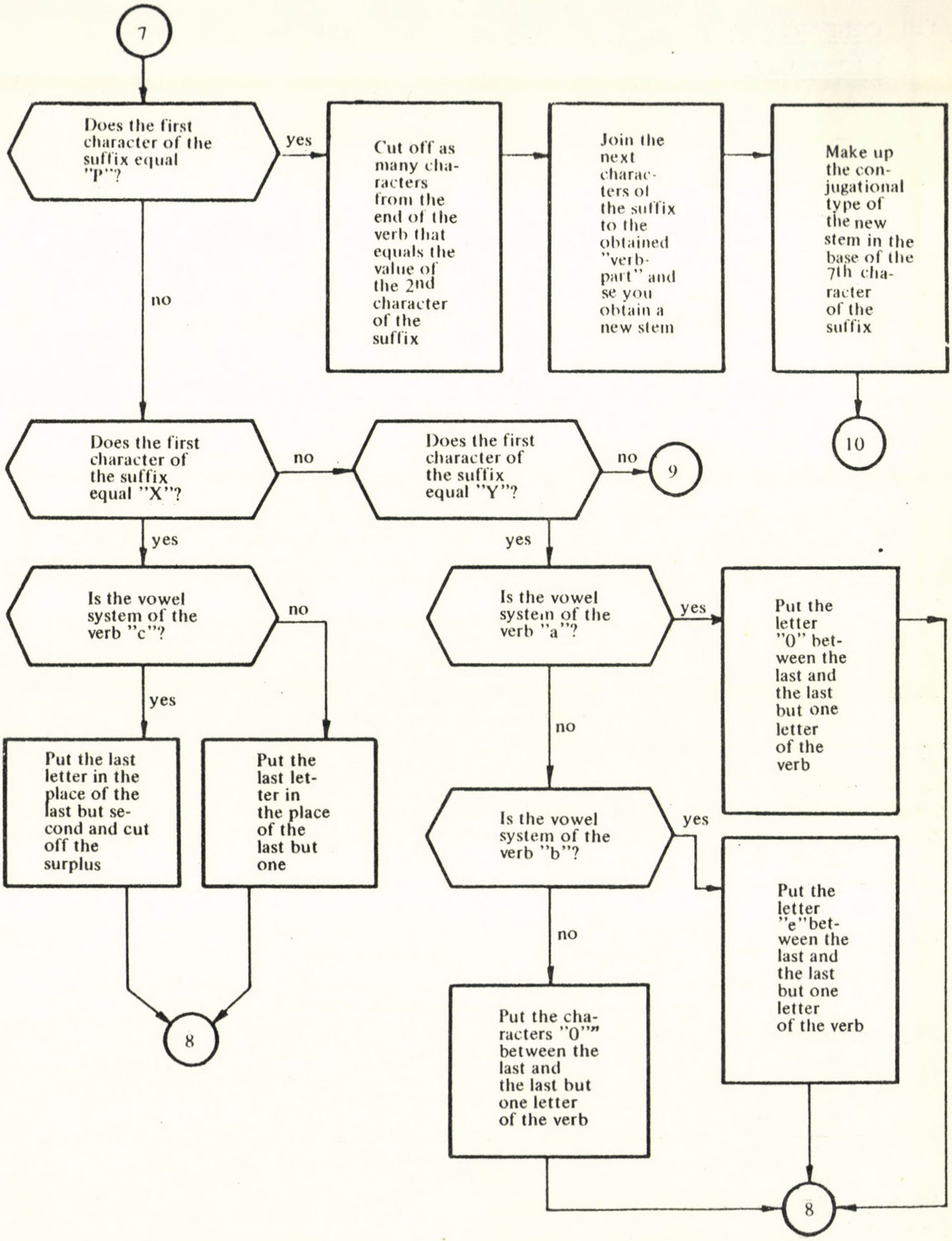
* The data given here do not correspond with the data given in para. 3.6.3, since in para. 3.6.3 the useful data storage was counted, only, but the data storage which is still necessary for filehandling was not taken into account.


1
Decide the code number of the demanded form
Can the demanded form exist?
no
Print: "SUCH A FORM DOES NOT EXIST"
STOP
yes
Search the verb in the dictionary
Is it found?
no
Has the verb a verbal suffix?
yes
no
Search the verb without its verbal suffix in the dictionary
Has the verb a characte-ristic termination?
yes
2
yes
Is it found?
no
no
yes
Search its type in the table of the charac-teristic terminations
Is it a lexeme?
Store its conjugational type
yes
no
2
3
Conjugate the deman-ded forms of the le-xemes one after the other
Print: "THE VERB MAY NOT BE FOUND IN THE DICTIONARY"
Print the obtained verbal forms
STOP


3
Was 1st person singular of a transitive verb demanded and do both forms exist?
yes
Make and print the form relating to an object of 2nd person
Make and print the form relating to an object of 3rd person
STOP
no
Was an infinitive with personal suffix demanded?
yes
Make the infinitive of the verb
Add the appropriate personal suffix
Print the conjugated form
STOP
no
Is there a systematical remark, i.e. a6 =∪?
yes
Does the demanded verbal form exist?
no
Print:"THE VERB LACKS THIS VERBAL FORM"
STOP
yes
no
Is the demanded form rare?
no
yes
Print:"RARE"
Is the verb a double conjugational one?
yes
Conjugate the appropriate form of the 1st verb
Conjugate the appropriate form of the 2nd verb and write it after the 1st verb
Print the result
STOP
10
no
Is the verb only used as a transitive?
yes
Cut the last letter of the verb
no
Has the verb the ending '-ik'
yes
Cut the two last letters of the verb
no
4


4
Search the record appropriate to the conjugational type of the verb on the DISELT file
Is the value of the bit corresponding to the code number in the record equal 1?
yes
Search the suffix number of the demanded form in the corresponding record of the DISMIN file
5
no
Is the conjugational type of the verb related to a standard type?
yes
Is the vowel system of the verb "a"?
yes
Search the suffix number of the demanded form in the record with the key "1a" on the DISMIN file
no
Search the suffix number of the demanded form in the record with the key "1b" on the DISMIN file
5
no
Is the conjugational type of the verb related to the conjugational type "URES"?
yes
Print:"THE VERB LACKS THIS VERBAL FORM"
STOP
no
Take note of the new conjugational type
5
Has the obtainad suffix number a verbal form?
no
Search the corresponding suffix number on the DISRAG file
yes
6
Conjugate the verbal forms corresponding to these suffix number one by one
Print the obtained forms
STOP


6
Is the verbal form "rare"?
yes
Calculate the suffix number the corresponding frequent suffix
Print: "RARE"
no
Search the suffix on the DISRAG file corresponding to the suffix number
Is the first character of the suffix not O,i.e. may the verbal form exist?
no
Is there any remark denoted in the suffix?
no
Print:"THE VERB LACKS THE DEMANDED FORM"
STOP
yes
Does it concern the demanded form?
no
yes
yes
no
7
Is the first character of the suffix a number?
yes
Cut off the characters (corresponding to the number) from the end of the verb
8
Cut off the first character of the suffix
9
Is there any remark denoted in the suffix?
yes
Does the remark concern the demanded form of the demanded verb?
yes
Form the new suffix number
5
no
no
Join the stem of the verb with the suffix
Cut off the last character from the end of the suffix
Print the obtained form
Is there any remark concerning the demanded form?
yes
Print the remark under the verb
no
STOP


7
Does the first character of the suffix equal "P"?
yes
Cut off as many characters from the end of the verb that equals the value of the 2nd character of the suffix
Join the next characters of the suffix to the obtained "verb-part" and se you obtain a new stem
Make up the conjugational type of the new stem in the base of the 7th character of the suffix
10
no
Does the first character of the suffix equal "X"?
no
Does the first character of the suffix equal "Y"?
no
9
yes
Is the vowel system of the verb "c"?
no
yes
Put the last letter in the place of the last but second and cut off the surplus
Put the last letter in the place of the last but one
8
yes
Is the vowel system of the verb "a"?
yes
Put the letter "0" between the last and the last but one letter of the verb
no
Is the vowel system of the verb "b"?
yes
Put the letter "e" between the last and the last but one letter of the verb
no
Put the characters "0"" between the last and the last but one letter of the verb
8

4. ADVANTAGES OF EL'S SYSTEM AND THE MACHINE SYSTEM

4.1 COMPARISON OF EL'S SYSTEM AND THE "EXPLANATORY DICTIONARY"

If we give the paradigmatic features of the entries of the Concise Explanatory Dictionary of Hungarian, in an analogous way to those of the Explanatory Dictionary [3] then for the description of the conjugational system twice as much place would be necessary than for the description of the full variant and 17 times more place than for the description of the contracted system as shown in Table 3. /The data of Table 3 must be considered approximate being set up in 1971, and since that time the system has slightly been modified. This modification, however, is not more than 1-2%./

Table 3

Size		The groups of the verbs without '-ik'	The groups of the verbs with '-ik'	Together
n-size using the notation of the Explanatory Dictionary /1/		104810	97975	202785
full variant	number of entries /2/	5600	2354	7954
	n-size /4/	13600 + 30409	8379 + 32765	21979+ 63174
n-size of the contracted system /1,2,5/		≈ 22000	≈10000	≈32000
con-tracted system	number of entries /3/	1514	392	1906
	n-size /5/	≈ 8000	≈4000	≈12000

The number of letters, necessary to write a word /text/ is called 't h e n - s i z e o f a w o r d /t e x t/.

/1/ In this variant the number of the entries equals the number of the entries of the full variant.

/2/ Verbs with a typical termination are not taken into consideration here.

/3/ Verbs with a typical termination described by the contracted system are taken into consideration.

/4/ The first number is the sum of the paradigmatic marks to be indicated in the dictionary and the second one is the n-size in the description of the conjugational types.

/5/ Here the n-size is only the sum of the paradigmatic marks to be indicated in the dictionary; the size of the description of the conjugational types /about 3/4 printed sheet/ is to be added [1]/.

EL's system gives a more exact description of the conjugational system than the "Explanatory Dictionary".* Namely, the latest gives only 2-3 characteristic verbal forms and the forms that differ from the forms which can be concluded from the indicated forms. At the end of the entries only the most frequent derivations may be found and the forms 56,59,63 of the verb are usually not published, and neither is their lack indicated.

4.2 COMPARISON OF EL'S SYSTEM AND THE MACHINE SYSTEM

There was a possibility to set up several solutions for the recursion between certain conjugational types.

1. If we described the conjugational types in an analogous way to EL's system then 7909 forms without '-ik' and 5341 forms with '-ik', altogether 13258 forms would differ from the standard types.

---

* The Explanatory Dictionary, however, does not aim to give an exact conjugational system.

2. If we use recursion as often as possible, then

   (i) denoting the elision as in para. 3.4.2.2, the number of divergences is 3252+2468=5720;
   (ii) denoting the elision as in para. 3.4.2.2, the number of divergences is 3179+2468=5467; while
   (iii) not considering the rareness and the order of the forms and denoting the elision as in para. 3.4.2.3, the number of the divergences is 2912+2186=5098.

3. If we only allow simple recursion, in the case (i) the number of the divergences is 4194+3704=7898, in the case (ii) it is 4150+3704=7854 and in the case (iii) it is 3888+3530=7418.

Considering all the solutions, only that one described in 2(ii) was used in the program.

Thus, using the recursions only half as much divergences must be denoted than in EL's system. The conjugational types may be looked over easily in the tabular form, but this description /on paper/ occupies twice as much place than EL's system.

5. APPLICATION OF THE PROGRAM AND THE DATA FOR

5.1 ANALYSIS

If we wish to solve the automatic analysis of verbs /e.g. in a translation program from Hungarian into another language/, we must use the detailed system and the program in order to recognize all the possible verbal forms. However, we need not take into account the rareness and order of the verbal forms and other stylistic remarks.

5.2 MACHINE TRANSLATION

In a program which translates into Hungarian from another language, it is enough to use the less detailed system i.e. for one code number it is enough to formulate one /namely the most frequent/ form. This saves 8K bytes.

If the machine translater program translates a given /special/ text, the DISTAR-file contains less verbs depending on the character of the text and then it is not necessary to insert the data of the conjugational types containing only 1 verb.

5.3 SOLVING LINGUISTIC PROBLEMS

There are several problems proposed by linguists which might be solved on the basis of the system /with the help of further programs/.

A/ In which conjugational types /verbs/ does the first person plural present tense in the indicative form of a transitive verb /code number 11/ differ from the imperative form /code number 51/?
/This is the well-known "suk-sük" problem which is important from the point of language-culture./
E.g. "olvassuk" /'we read'/: 1st person plural present tense in the indicative form of a transitive

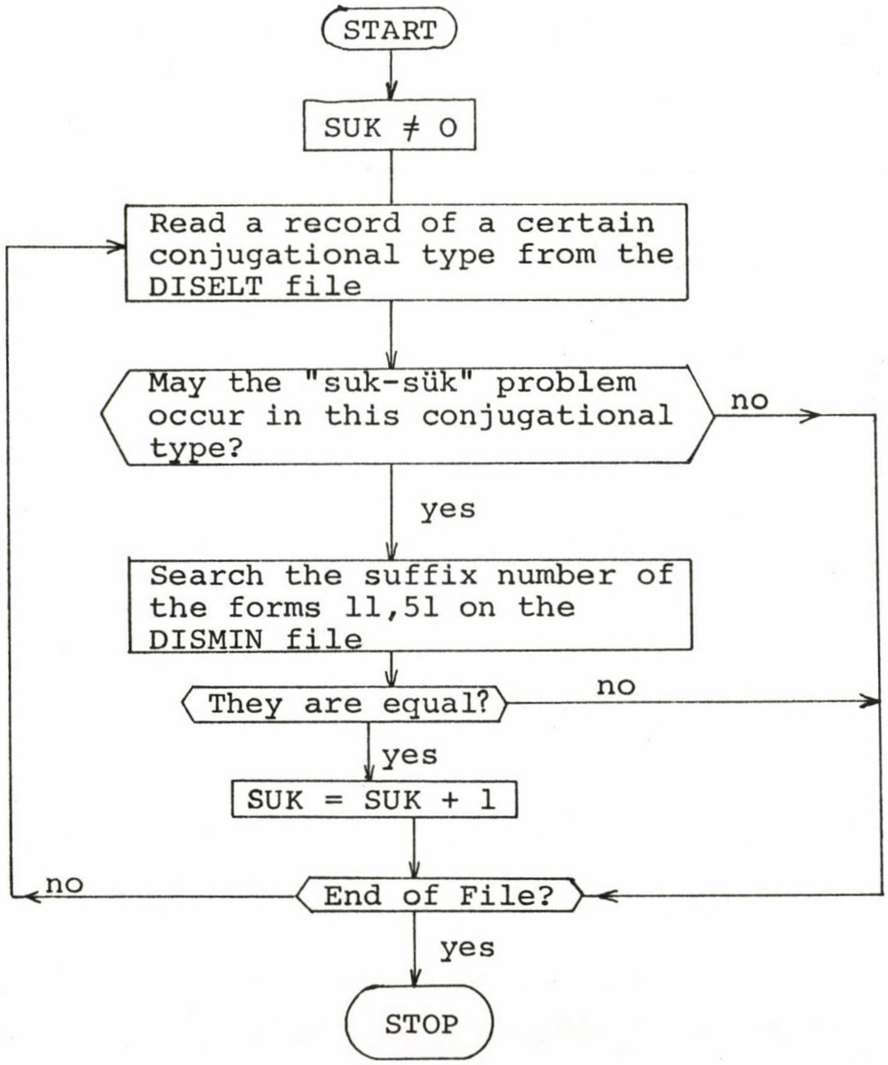
verb; and

"olvassuk" /'let us read'/: 1st person plural present tense in the imperative form of a transitive verb; BUT

"avatjuk" /'we initiate'/: 1st person plural present tense in the indicative form of a transitive verb; and

"avassuk" /'let us initiate'/: 1st person plural present tense in the imperative form of a transitive verb.

This would be a plan of a program that solves this problem:



In this way we may get known in how many conjugational types there is a difference between the 2 forms. If we want to know the number of such verbs then we have to make the program remember the conjugational types in the case of equal suffix number. After this we examine the number of the verbs belonging to such a conjugational type with the help of the DISTAR file. The number of the verbs belonging to group III /see para. 3.8.1/ is given by the following formula:

$$A = \sum_{\substack{\text{conjuga-}\\\text{tional}\\\text{types}}} \text{number of verbs belonging to the conjugational type}$$

The number of the verbs belonging to group II and for which the 2 suffix numbers are equal, is given by the following formula:

$$B = \sum_{\substack{\text{typical}\\\text{termination}}} \text{number of verbs with a certain typical terminations}$$

If we take into account that the number of the verbs belonging to group I /about/ equal to the number of the verbs belonging to group II and III, then the "suk-sük" problem refers to about 2/A+B/ verbs.

B/ In the case of the verbs with '-ik' how many of them do necessarily take 'ik' and which may also be used without '-ik'?

C/ Is the form variant of the stem a morphological variant or an orthographical one and which is the more frequent one?

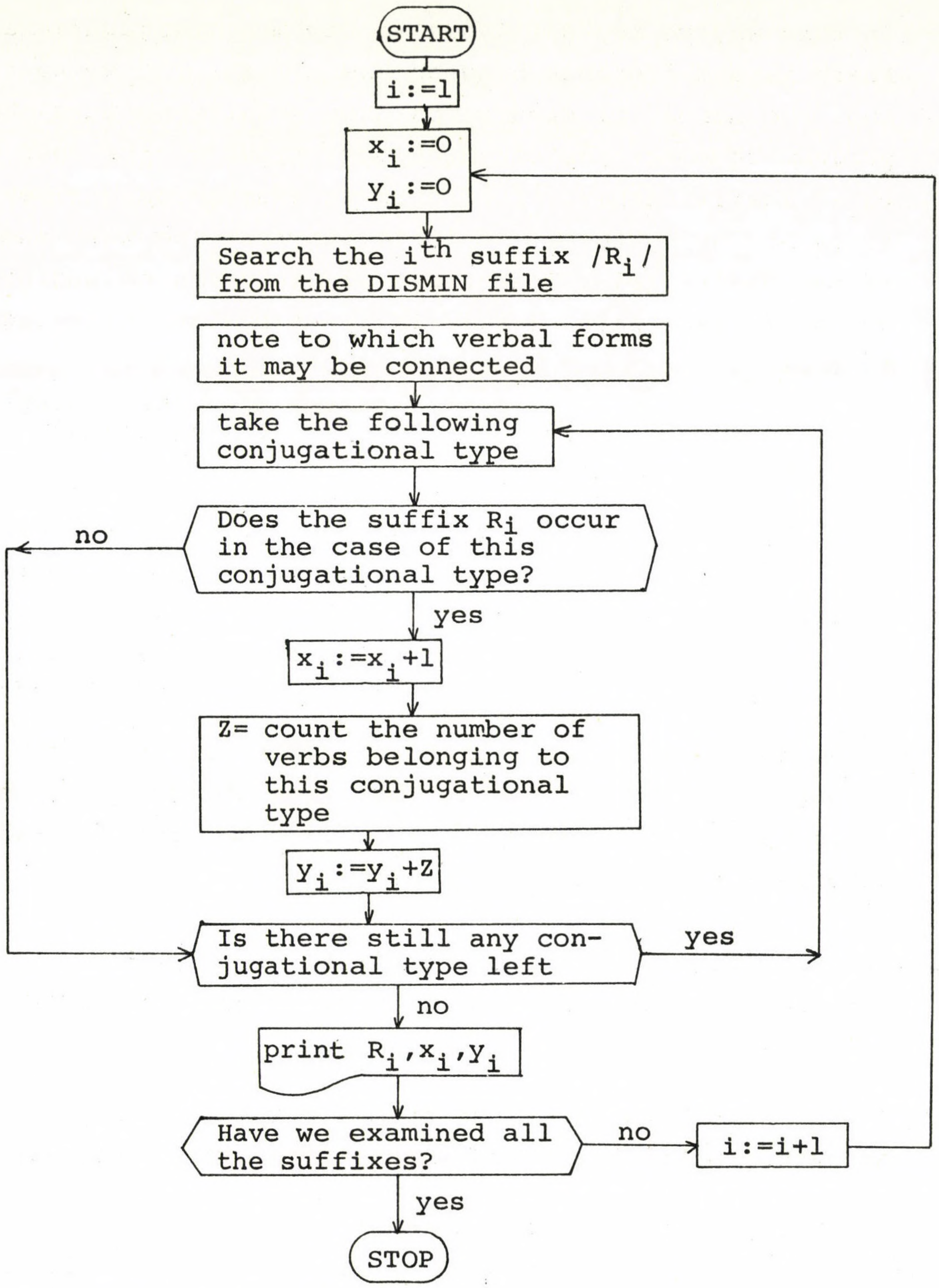
E.g. Morphological stem variant:

"avat" - "avassak" /'he initiates' - 'let me initiate'/;

Orthographical variant:

"fogódzik" - "fogóddzam" /'he clings' - 'let me cling'/

D/ In how many conjugational types /verbs/ does a suffix occur? A flow-chart to solve this problem may be the following:



Legend: i = suffix number

R_i = the suffix which belongs to the 'i'th suffix number

x_i = the ith suffix occurs in x_i conjugational types

y_i = the ith suffix occurs in y_i verbs

Z = auxiliary variable

E/ The system may be used to solve certain designation problems.

E.g. we wish to determine which is the more typical one of the 2 suffixes /A and B/ of the same verbal form, then we should examine, how often the suffixes occur in the conjugational types /X_A, X_B/. If the suffix A occurs more often than B, i.e. $X_A \gg X_B$, A is called typical and this is denoted by the sign "+". If $X_A \approx X_B$, the productivity of the suffixes A and B should be examined, namely number of verbs taking suffix A and the number of verbs taking suffix B and the problem should be solved on this basis.

E.g. We should like to decide which is the typical suffix of those of the form 62 of the verbs belonging to the vowel system "a". We found that suffix "-al" occurs once, suffix "-aszt" occurs 6 times, suffix "-at" occurs 34 times, suffix "-it" occurs 4 times, suffix "-lal" occurs once, suffix "-kat" occurs once, suffix "-t" occurs 2 times, suffix "-tat" occurs 78 times, suffix "-vat" occurs twice and all the 64 conjugational types lack the form 62. Thus suffix "-tat" is considered to be the typical one and this is the suffix of the standard types la too /i.e. if this suffix occurs in a conjugational type, we do not give the form 62/.

I. APPENDIX

A/ Original variation /with EL's numbering/

[1a ápol] /'nurse'/ 1~ok, 2~sz, 3~, 4~unk, 5~tok, 6~nak, 7~lak;
8~om,~od, 9~ja, 10~juk, 11~játok, 12~ják;
13~tam,~tál, 14~t, 15~tunk,~tatok, 16~tak;
17~talak,~tam,~tad, 18~ta, 19~tuk,~tátok, 20~ták;
21~nék, 22~nál, 23~na, 24~nánk,~nátok, 25~nának;
26~nálak,~nám,~nád, 27~ná, 28~nánk v.~nák,~nátok, 29~nák!
30~ni! 31~jak, 32~j/~jál/, 33~jon, 34~junk,~jatok, 35~janak,
36~jalak,~jam, 37~d/~jad/, 38~ja, 39~juk,~játok,~ják!
40~ó, 41~t, 42~andó! 43~va, 44~ván! 45~hat, 46~ható!
47~ás! 48~tat, 49~tatik

[1a5 durran] /'explode'/ As 1a but only intransitive[3]:
42,46:--! 48~t

[1a8 buggyan] /'rise'/ As 1a but only 3.person intransitive[3]:
42,46:--! 48~t

[1b emel] /'lift'/ 1~ek[ë], 3~, 4~ünk, 5~tek[ë], 6~nek;
7~lek, 8~em[ë],~ed[ë], 9~i, 10~jük, 11~itek[ë], 12~ik;
13~tem,~tél, 14~t, 15~tünk,~tetek[e-ë], 16~tek;
17~telek,~tem,~ted, 18~te, 19~tük,~tétek[ë], 20~ték!
21~nék, 22~nél, 23~ne, 24~nénk,~nétek[ë], 25~nének;
26~nélek,~ném,~néd, 27~né, 28~nénk v.~nők,~nétek[ë], 29~nék!
30~ni! 31~jek, 32~j/~jél/, 33~jen[ë], 34~jünk,~jetek[e-ë]
35~jenek, 36~jelek,~jem, 37~d/~jed/, 38~je, 39~jük,
~jétek[ë],~jék!
40~ő, 41~t, 42~endő! 43~ve, 44~vén! 45~het, 46~hető!
47~és, 48~tet, 49~tetik

---

[3] Taking into account the dialectical variant of the following verbs: "bukkan" /'strike upon'/, "csattan" /'clap'/, "durran" /'explode'/, "kibuggyan" /'spout'/, "koppan" /'sound'/, "lobban" /'flare up'/, "nyikkan" /'squeak'/, "pattan" /'crack'/, "pottyan" /'plump'/, "villan" /'flash'/, "torpan" /'stop dead'/ : 3~/~ik/

[3a1 csikland] /'tickle'/ As 1a, but 2~asz/~sz/, 5~tok v.~otok, 6~anak, 7~alak /~lak/, 13, 15-20:~tam /~ottam/ etc., 14, 41:~ott! 21-30:~anék etc.! 42:-! 48-49:~oztat etc.[2]

[3b1 kérd] /'ask'/ As 1b, but: 2~esz, /~sz/, 5~etek[ë-ë] 6~enek, 7~elek, /~lek/, 13,15-20:~ettem v.~tem etc., 14-41~ett[ë], 21-30:~enék etc.! 48-49: /~eztet[ë-ë]

[4b3 ért] /'perceive'/ As 1b, but: 2~esz/~sz/, 5~etek [e-ë], 6~enek, 7~elek, v./~lek/, 13-20:~ettem[e-ë]etc.! 21-30:~enék etc.! 31-39:~sek etc.! 41~ett[ë],! 48-49:~et etc.

[5b2 téveszt] /'miss the target'/ As 1b, but: 2~esz, 5 etek[ë-ë], 6~enek, 7~elek, v.~lek, 13-20:~ettem [ë-ë] etc.! 21-30:~enék etc.!

[5c2 fürösztt] /'bathe sy'/ As 1b, but: 1~ök, 2~esz, 5~ötök, 6~enek, 7~elek v.~lek, 8~öm,~öd, 13-20:~öttem etc. 21-30:~enék etc.! 31-32, 34-39:~sszek etc., 33~sszön! 41~ött! 48-49:~et etc.

[5c3 föst]/'paint'/ As 5c2, but: 31-39:~ssek etc.

[6a1 bomol] /'resolve into'/ As 1a, but only intransitive: 1,4:.mlok etc. 2~sz/..mlasz/,; 14~t /..mlott/! 21-30:~nék /..mlanék/ etc. 40,42,47:..ló etc., 41..mlott v.~t

[7a1 tipor] /'trample'/ As 1a, but: 1,4,8:..prok etc., 2~sz/..prasz/, 6~nak/..pranak/, 14,41:~t/..prott/! 21-30:~nék/..pranék/ etc.! 40,42,47:..pró etc.! 48-49:~tat v. ...prat etc.

[9a elvan] /'be away'/ As 1a, but only intransitive: 1..agyok, 2..agy, 4..agyunk, 5..agytok, 13-16:..voltam etc.! 21-25...volnék etc.! 41/..volt/, 42:-!! 30-35, 40, 43-48: instead of tese: ellesz 9b1

---

2) the different forms of the verb "mond" /'say'/: 42~andó, 48-49:~at etc.

[10b2 nincs] /'there is no'/ only 3~ v. ~en, 6~enek

[11a gyónik] /'confess'/ As 1a, but: 3~ik/~/[2], 48~tat/~at/[1], 49/~atik/

[12a4 mosakodik]/'wash'/ As 1a, but only intransitive: 1/~om/, 2/~ol v. ~sz/, 3/~ik/, 14,41:~ott! 21~nék v.~nám, 23~na/~nék/! 31~jam v. ~jak, 33~jék v. -~jon! 46: -! 48/~tat/!! /instead of the forms 1-4 preferably ..kszik 19a/

[19a furakszik] /'push'/ 1~om, 2~ol, 3~ik, 4~unk, 5/~otok, v. ..kosztok/, 6~anak/..kosznak/! 7-48:- /instead of these: ..kodik 12a4/

[19a4 nyugszik] /'take a rest'/ As 1a, but only intransitive: 1~om, 2~ol, 3~ik, 5..gosztok v. ~otok /..godtok/, 6~anak v. ~gosznak /..godnak/, 13..godtam etc., 14..godott, 15-16:..godtunk etc. 21..godnék v. ..godnám, 22..gondnál, 23..godna v. ..godnék, 24-25, 30: ..godnánk etc.! 31..godjam v. ..godjak, 32..godjál v. ..godj, 33..godjék v. ..godjon, 34-35: ..godjunk etc.! 40..gvó, 41..godott /..godt/! 43-44:..godva etc.! 45..godhat /..ghat v. ..ghatik/! 47..gvás! 48..gtat! 42-46: -

---

1 with a difference of meaning

2 without '-ik' especially in transitive usage

B/ The transformed, variant

Conjugational type		1a			1a5		
Example		ÁPOL			DURRAN		
To which type it was related in EL's system					1a		
form number	code number	suffix	recursional difference	suffix num.	suffix	recursional difference	suffix num.
1	1	~ok	1	Ø4Ø			
2	2	~sz	1	Ø51			
3	3	~	1	ØØ1	~8	1	Ø14
4	4	~unk	1	149			
5	5	~tok	1	144			
6	6	~nak	1	111			
7	7	~lak	1	1Ø6			
8	8	~om	1	Ø42			
	9	~od	1	Ø39			
9	10	~ja	1	Ø31			
10	11	~juk	1	1Ø5			
11	12	~játok	1	8Ø9			
12	13	~ják	1	251			
13	14	~tam	1	538			
	15	~tál	1	299			
14	16	~t	1	ØØ9			
15	17	~tunk	1	315			
	18	~tatok	1	554			
16	19	~tak	1	137			
17	20	~tál	1	552			
18	21	~tam	1	138			
	22	~tad	1	136			
	23	~t	1	153			
19	24	~tuk	1	147			
	25	~tátok	1	844			
20	26	~ták	1	298			
21	27	~ná	1	269			
22	28	~nál	1	266			
23	29	~na	1	Ø35			
24	30	~nánk	1	52Ø			
	31	~nátok	1	816			
25	32	~nának	1	815			
26	33	~nálak	1	814			
	34	~nám	1	265			
	35	~nád	1	264			
27	36	~nál	1	11Ø			
28	37	~nánk/~nók/	1	52Ø			
	38	~nátok	1	816			
29	39	~nák	1	265			
30	40	~ni	1	Ø37			

Conjugational type		1a			1a5		
Example		ÁPOL			DURRAN		
To which type it was related in EL's system					1a		
form number	code number	suffix	recursional difference	suffix num.	suffix	recursional difference	suffix num.
31	41	~jak	1	Ø97			
32	42	~j/~jál/	1	ØØ6			
33	43	~jon	1	1Ø4			
34	44	~junk	1	262			
34	45	~jatok	1	512			
35	46	~janak	1	511			
36	47	~jalak	1	51Ø			
	48	~jam	1	Ø98			
37	49	~d/~jad/	1	ØØ3			
38	50	~ja	1	Ø31			
39	51	~juk	1	1Ø5			
	52	~játok	1	8Ø9			
	53	~ják	1	251			
40	54	~ó	1	Ø38			
41	55	~t	1	ØØ9			
42	56	~andó	1	494	-	1	ØØØ
43	57	~va	1	057			
44	58	~ván	1	317			
45	59	~hat	1	Ø91			
46	60	~ható	1	5Ø7	-	1	ØØØ
47	61	~ás	1	Ø77			
48	62	~tat	1	139	~t	1	ØØ9
49	63	~tatik	1	553			
recursional conjugational type	64		Ø			1	
systematical remark					only intransitive		

statistic part	Number of verbs regular termination	-ál -al -ol	753 50 475	
	other termination	-an	32	15
	Total amount		1310	15
	Regular conjugation, omissible paradigme number	-al -ál -an -ol -ül	50 753 2 474 14	15
	Total amount		1273	15
	the paradigme numbers which must be indicated		17	0
Remark type III.	8/ taging into account the dialectical variant of the following verbs "bukkan" /'strike upon'/, "csattan" /'clap'/, "durran" /'explode'/, "kibuggyan" /'spout'/, "koppan" /'sound'/, "lobban" /'flare up'/, "nyikkan" /squeak'/, "pattan" /'crack'/, "pottyan" /'plump'/, "villan" /'flash'/, "torpan" /'stop dead'/: 3~/~ik/			

II. APPENDIX

THE TABLE OF THE TYPICAL TERMINATIONS

Termination	Conjugational type	Number of verbs	Number of exceptions
-ad	1a1	39	43*
-al	1a	50	5
-ál	1a	753	25
-all	3a3	10	6
-an	1a1	25	35*
-ant	4a6	27	11
-ász	4a4	23	--
-ászik	15a4	26	1
-aszt	5a3	62	4
-at	5a	229	108
-az	4a	56	1
-áz	4a	146	28
-ázik	14a1	125	38
-dös	4c3	7	--
-edik	12b	234	133
-eg	2b	41	53*
-el	1b	226	49
-él	1b	49	13
-en	1b5	23	25*
-eng	3b	11	7
-ent	4b6	20	9
-es	4b3	7	2
-ész	4b4	10	--
-észik	15b4	7	4
-eszt	5b3	45	1
-et	5b	177	77
-ez	4b	175	45
-éz	4b	28	2
-ezik	14b	24	17
-od	2a	6	--
-odik	12a	380	132
-ódzik	15a	14	18*
-og	2a	86	94*
-ol	1a	474	56
-ong	3a	21	--
-os	4a3	16	1
-oz	4a	276	43
-óz	4a	43	5
-ozik	14a1	105	95
-ózik	14a1	30	48*
-öd	2c	6	--
-ödik	12c	45	40
-ődik	12c	65	54
-ődzik	15c	8	1
-ög	2c	29	26

* The exceptions are from several conjugational types

Termination	Conjugational type	Number of verbs	Number of exceptions
-öl	1c	76	14
-öz	4c	30	9
-őz	4c	12	3
-özik	14c5	4	11*
-őzik	14c	14	19*
-ul	1a1	123	31
-ül	1c1	135	15

* The exceptions are from several conjugational types

BIBLIOGRAPHY

[1] Elekfy, L.: A magyar szóvégek és toldalékok rendszere. /The System of the Hungarian Word Endings and the Suffixes./ /in: Magyar Nyelv, LXVIII.1972. 303-309 and 412-429 pp./

[2] Elekfy, L.: Szókincsünk nyelvtani alakrendszere. /Grammatical Form System of Hungarian Word-Stock/; /Typescript/

[3] Értelmező Szótár: I.-VII.kötet /Explanatory Dictionary, Vol.I.-VII./

[4] Kelemen, J.: Beszámoló a gépi nyelvstatisztikai kérdésekről /a debreceni nyelvészkongresszus előadásai/. /Report on Some Questions of the Computer Linguistic Statistics - Proceedings of the Debrecen Linguistic Congress./ Ed.by S.Imre and I.Szatmári. Budapest, Akadémiai Kiadó, 1966.p. 480-485./

[5] Magyar Értelmező Szótár. /Concise Explanatory Dictionary of Hungarian/ Budapest, Akadémiai Kiadó, 1972.

[6] Máthé, J. - Kovács-Bölöni, E. /Mrs/ - Schweiger, P. - Székely, E.: A magyar igeragozás független analizisének egy modelljéről /a debreceni nyelvészkongresszus előadásai/. /About one Model of the Independent Analysis of the Hungarian Conjugational System - Proceedings of the Debrecen Linguistic Congress./ Ed. by S.Imre and I.Szatmári. Budapest, Akadémiai Kiadó, 1966. p. 499-502.

[7] Papp, F.: Algoritmus. /Algorithm/ /in: Magyar Nyelvőr 89., 1965. p.87-93./

AN ALGORITHM FOR FINITE GALOIS-CONNECTIONS

G.FAY
Institute for Economy Organisation and
Computational of Metallurgy and Engineering Industry
Budapest, Hungary

1. INTRODUCTION

The many-to-many relationships between things in practice /rather than one-to-ones as usually considered in applications dealing with e.g. numerical functions/ give rise to the question how to, so to speak, "represent" a many-to-many correspondence in possibly as convenient a form as is customary in everyday applications concerning ordinary functions.

A possible algorithm is given here for reducing many-to-many mappings of finite sets to one-to-ones.

This is a practical way to produce Galois-connection between two finite sets and also to determine all the substructures of a certain algebraic structure. The lattice theoretical preliminaries can be found in Szász /1963/, where further references concerning Galois connections are available. In our paper, however, an effort is made to be fairly self-contained.

From linguistical points of view this paper is motivated by an observation of N.Chomsky and M.P.Schützenberger /1961/ who wrote, "... it is possible that general questions concerning the formal properties of context free systems and formal relations between them may have a concrete interpretation in the study of data processing systems as well as in the study of natural language."

Now it is clear that no natural language can dispense with notions. Intuitively a notion is not simply a feature or a property which is possessed by a set of things. It is, rather a set of properties whose each member is possessed by every member of a set of things. The notion of "dog" must contain all the features which are possessed by all the dogs. So, in other words, the concept of notion should be richer than the concept of a set. It is not a set, but rather a pair of sets.

Where do we get notions from? How do they get into our language? It seems that is does, through a procedure described by J.E.L.Farradane /1966/ which, in turn, from mathematical point of view, looks like leading to the form of a closure operation. Gathering observations from the nature man /or rather child/ step by step builds up the sets of objects and the sets of "features" with a relation such that all the objects /"things"/ of the set possess all the features /"properties"/ of the latter set.

On the other hand, confronted with the "artificial nature" /or with artifacts/ - and this is what we are concerned with in data processing - one cannot dispense with the notions abstracted from the data unless one, wants to be lost in the chaos of informations.

How to aid the procedure of "conceptualization" of the bare sets of data in order to incorporate the artifactual notions in our artificial language to be developed?

No doubt, first, the characteristics of the concept of concept is needed, second, an algorithm to produce them is highly desirable.

The author is completely aware of the problem of notion concept belonging to the fields of Symbolic Logic, and that R.Carnap

/1942/ and Y.Bar-Hillel /1964/ extensively dealt with these kind of problems, To my knowledge, however, there is no theoretical approach which tacles the question of "conceptualization" applying techniques based on the theory of Galois-connections. This paper tries to do this.

A couple of years ago a somewhat similar approach has been made for "conceptualizing observations". In 1970 /Fay, 1970/ I called this procedure "essentialization". This effort was motivated by quantum logic whose study is highly recommendable to those longing for refreshing ways of thinking in mathematical linguistics.

It seems to me, that in computer science, characteristically, only logical inferences are attempted to be implemented in machines. What we really need, however, is to extend-aided-by computers our ability to make factual inferences. We are not short of rules like "All men are mortal, Socrates is man so Socrates is mortal". We rather badly need rules of inference like "if an animal is mammal, then is has no wings."

2. GALOIS CONNECTIONS AND CLOSURE OPERATIONS

Let U and V be any two sets and ϕ is a relation defined on the product set UxV. If for a pair u,v($u \in U$, $v \in V$) Φ holds, we write /as usual/ $u\phi v$ or $v\phi^{+}u$. Define for any $u \in U$, $v \in V$

$$\phi(u)=\{v|u\phi v\} \qquad (\subseteq V)$$

and /dually/

$$\phi^{+}(v)=\{u|v\phi^{+}u\} \qquad (\subseteq U)$$

Further, for any $X \subseteq U$, $Y \subseteq V$ we have by definition

$$\phi(X) = \bigcap_{x \in X} \phi(x), \quad \phi^{+}(Y) = \bigcap_{y \in Y} \phi^{+}(y)$$

and

$$\varphi(X) = \phi^{+}(\phi(X)) \qquad (\subseteq U)\ ;$$

$$\varphi^{+}(Y) = \phi(\phi^{+}(Y)) \qquad (\subseteq V)\ .$$

Now the following facts are well-known /see e.g. Szász /1963/ p. 70-71/.

1. Mappings

$$\varphi : SbU \to SbU\ , \quad SbV \to SbV$$

/SbU=set of all the subsets of U/ are <u>closure</u> operations of the class of all the subsets of U and V, respectively. We say that closures φ, φ^{+} are <u>induced</u> by the relations ϕ, ϕ^{+} /or simply ϕ induces φ/.

2. Let L, L^{+} be the sets of all the φ-closed, φ^{+}-closed sets of U, V, respectively. One can introduce lattice operations on L and L^{+} with repsect to the ordering $\subseteq$ as

$$\begin{aligned} a \cup b &= \inf\{a,b\} \\ a \cup b &= \sup\{a,b\} \end{aligned} \qquad \text{for either } a,b \in L, \text{ or } a,b \in L^{+}$$

Both, the structures $\underline{L}=\langle L,\cap,\cup\rangle$ and $\underline{L}^{+}=\langle L^{+},\cap,\cup\rangle$ are /complete/ lattices. /Clearly $L \subseteq SbU$, $L^{+} \subseteq SbV$./

3. If $X \subseteq U$, $Y \subseteq V$, X and Y is <u>closed</u> i.e.

$$X = \varphi(X) \quad \text{and} \quad Y = \varphi^{+}(Y)$$

then the mappings $Y = \phi(X)$, $X = \phi^{+}(Y)$

$$\underline{\phi} : \underline{L} \to \underline{L}^{+}\ , \quad \text{and} \quad \underline{\phi}^{+} : \underline{L}^{+} \to \underline{L}$$

are both dual isomorphisms with respect to the set theoretical inclusion. This pair of dual isomorphisms is said to be the

<u>Galois connection</u> /between the sets U,V with respect to the relation ϕ/, Given lattices $\underline{L}$, $\underline{L}^+$ one can form the set of all the pairs

$$<a,\phi(a)> \ , \ a\in L, \quad \phi(a)\in L^+$$

Now $\underline{a}$, as a closed set of things /records, rows of a table etc./ tpgether with $\phi(a)$ can be interpreted as a <u>notion</u> or a "conceptualized representative of a collection of data". As for $\phi(a)$ as a set of y's they can play the role of a collection of properties or attributes all of which all the things belonging to $\underline{a}$ /$\underline{a}$ is a set!/ possess. The dual lattice theoretic structure of the set $\{a,\phi(a)|a\in L\}$ enables as to develop a kind of a "data logic". Take e.g.

$$a = \{u_1,u_3,u_4\} \ , \ b = \{u_1,u_3,u_4,u_7\} \ ,$$

$$\phi(a)=\{v_{14},v_{15},v_{17},v_{18}\} \ , \qquad \phi(b)=\{v_{14},v_{15},v_{18}\}$$

Being $a\subset b$, we say: "every a is b", $\phi(a)$ being a <u>common</u> feature of the a's $\phi(b)$ of b's, we can <u>infer</u> from feature in the following way:

If a thing /record, entity, row, object/ possesses <u>any</u> of the attributes of the class $\phi(a)$ then it must possess <u>all</u> the attributes of $\phi(b)$. /Dont be misled by $\phi(a)\supset\phi(a)$./ This inference yields <u>some factual new</u> /c.f. Bar-Hillel 1952./ if we chose for a feature v_{17} and observe that in this /rather restricted/ world of data $\{u_1,u_3,u_4,u_7\}$,

$$v_{17} \ \underline{\text{factually implies}} \ v_{14},v_{15} \text{ and } v_{18} \ .$$

Of course, the question of putting together restricted /worlds of data/ files arises. By our algorithm, to present here, all these kinds of factual implications will easily be available. It seems that factual implications tell deeper features about the contant of data sets than the feeble manconceived queries. The relevance of semantic information theory has been very thoroughly dealt with by Bar-Hillel /1952/.

3. THE FINITE CASE

From now on let us suppose that both U and V are finite. In applications it is interesting how to actually construct lattices $\underline{L}$, $\underline{L}^+$ by the sets U, V and by the relation ϕ. By Szász /1963/ a few interesting applications can be found /p.72/, e.g. using these dual isomorphisms and the closure operation φ one can produce the basic theorem of Galois theory, some projective geometrical, group theoretical and number theoretical results.

The relevance of finiteness of the basic sets U and V is shown at the first place in the theory of data banks and information retrieval. /E.g. to a supplier there belongs many supplies and vice-versa; or projects and parts are usually in many-to-many relationships./

In the relational approach to data banks "conceptual" processing of data is quite at hand. The formal candidate of a concept is nothing else than a φ-closed set with respect to the relation ϕ in question.

Relational data base management systems are extensively studied at IBM San Jose /California/ centering around Codd /1969/.

Let we are given now the /finite/ sets U,V and the relation ϕ between their elements. In order to produce the lattice $\underline{L}$ of all the φ-closed sets of U one have to decide on whether a given subset X of U is closed or not. This of course cannot be done by a brute straitforward approach. For if U contains n elements then 2^n cases would have to be examined. And even in each case a couple of fairly complicated operations would be to carry out.

Viz., firstly one would form the sets $\phi(x)$ for all $x \in X$. Secondly to form the meets

$$\phi(X) = \bigcap_{x \in X} \phi(x)$$

Thirdly the sets $\phi^+(y)$ satisfying the condition, $y \in \phi(x)$ Fourthly one have to meet these sets together yielding

$$\phi^+(\phi(x))$$

Lastly one have to decide which of the relations

$$X \subset \phi^+(\phi(X)) \quad \text{or} \quad X = \phi^+(\phi(X))$$

holds.

Altogether these five steps would lead to at least five elementary operations, on principle one had to carry them out on all the subsets X of U and Y of V which would mean finally /in general/

$$(5 + 5)\ 2^n$$

instructions. /In case n=20 it is over ten million and in n=60 over 10^{19}./

4. U, V GENERATORS AND φ-CLOSED SETS

Consider two finite sets U and V with cardinality m and n respectively. Let

$$U = \{u_1, u_2, \dots, u_m\}\ , \quad I = \{1,2,\dots,m\}$$

$$V = \{v_1, v_2, \dots, v_n\}\ , \quad J = \{1,2,\dots,n\}$$

Let

$$R_\phi = \{\langle u_i, v_j \rangle \mid u_i \phi v_j,\ i \in I,\ j \in J\}$$

Clearly

$$R_\phi = \subseteq U x V$$

being the set theoretical representation of the relation. It can also be given in a tabular /matrix/ form. Arrange the elements of R_ϕ in an m-row n-column matrix and put a digit 1 /or a cross/ into hhe meet of the i-th row and j-th column whenever $u_i \phi v_j$ is the case and put a 0 /or blank/ otherwise. To the description of the algorithm for determining lattices $\underline{L}$ and $\underline{L}^+$ and mappings φ and φ^+ there will be attached an example whose data have been selected at random. See Table I.

Firstly consider row-vectors $\underline{u}_i = \underline{U}_i\{u_{i1}, u_{i2}, \ldots, u_{ij}, \ldots, u_{in}\}$ where

$$u_{ij} = \begin{vmatrix} 1 \text{ of } u_i \phi v_j \text{ ,} \\ 0 \text{ otherwise.} \end{vmatrix}$$

Similarly introduce column-vectors as

$$v_j^+ = v_j^+ \{v_{j1}, v_{j2}, \ldots, v_{ji}, \ldots, v_{jm}\}$$

with

$$v_{ji} = \begin{vmatrix} 1 \text{ if } v_j \phi^+ u_i \\ 0 \text{ otherwise} \end{vmatrix}$$

Clearly

$$v_{ij} = v_{ji} \text{ .}$$

We refer to u_{ij} and $v_{k\ell}$ as

$$u_{ij} = (u_i)_j$$

$$v_{k\ell} = (v_k^+)_\ell$$

Secondly introduce a

DEFINITION

A set $X \subseteq U$ /$Y \subseteq V$/ is called a U-generator /UG/ V generator, /VG/ iff there exists an element

$$v_X \in V \quad u_Y \in U$$

such that

$$X = \phi^+ (v_X) \ , \ (Y = \phi(u_Y)) \quad .$$

The subsets of U and V are stipulated to be called simply generators. We introduce the empty set O as U- or V-generators, too. Moreover for uniformity we speak of the noughtelement O_U and O_V of U and V, respectively, formally defined by

$$x \phi O_V \qquad \text{never holds}$$

$$O_u \phi y \qquad \text{never holds .}$$

We have now

$$\phi^+(\phi(O_V)) = O, \quad \phi(\phi^+(O_u)) = O$$

THEOREM 1

Every U-generator /V-generator/ is φ-closed /φ^+-closed/.

Proof: By symmetry reasons it is enough to consider the case of U-generator. If X is a UG then the element v_X /with $X=\phi^+(v_X)$/ clearly has the property that for each $x \in X$

$$\{v_X\} \subseteq \{y | x\phi \ y\} = \phi(x)$$

Therefore

$$\{v_X\} \subseteq \bigcap_{x \in X} \phi(x) = \phi(X)$$

By this we have

$$\phi^+(\phi(X)) = \bigcap_{y \in \phi(x)} \phi^+(y) \subseteq \bigcap_{y \in \{v_X\}} \phi^+(y) =$$

$$= \phi^+(v_X) = X \quad .$$

THEOREM 2

If X is a φ-closed set $/\subseteq U/$ then it is a meet of U-generators.

Proof: $$X = \phi^+(\phi(X)) = \bigcap_{y \in \phi(x)}$$

and, of course, every $\phi^+(y)$ is a U-generator.

THEOREM 3

The set theoretical intersection of two φ-closed sets is φ-closed.

Proof: By the closure property monotonity we have for any

$$X_1, X_2 \subseteq U \quad X_1 = \varphi(X_1) \ , \quad X_2 = \varphi(X_2)$$

$$X_1 \cap X_2 \subseteq X_1, X_2 \quad \text{implies} \quad \varphi(X_1 \cap X_2) \subseteq \varphi(X_1) \ , \varphi(X_2)$$

i.e.

$$\varphi(X_1 \cap X_2) \subseteq \varphi(X_1) \cap \varphi(X_2) = X_1 \cap X_2$$

while the opposite inclusion fulfils by the definition of the closure.

Combining Theorems 2 and 3 we have

THEOREM 4

A subset X of U is φ-closed if and only if Y is a meet of U-generators.

DEFINITION

The structures $\langle$UG,$\cap\rangle$, $\langle$VG,$\cap\rangle$ /closed under the set theoretical operation $\cap$ meet/ are called U-generator semigroup, UGS, and V-generator semigroup VGS, respectively. Clearly, the set of all the elements of UGS is $\varphi(U)$ and dually the set of all the elements of VGS is $\varphi^{+}(V)$.

We stipulate that every element of U is called UGS-generator, similarly $v \in V$ is VGS-generator. In VGS /VGS/ the algebraic operation "meet" is defined by the

DEFINITION

By a product / or lattice theoretic "meet"/ in symbol $\cap$ of two elements of UGS u_i and u_j we mean the set of all v-s which are in relation ϕ with both u_i and u_j. This set of v-s are sometimes written as u_k but with k>m:

$$u_i \cap u_j = u_k \quad \text{whenever } \phi(u_i) \cap \phi(u_j) = \phi(u_k) .$$

So, symbol $\cap$ means that the operands /a and b in $a \cap b$/ are considered as sets defined above.

In general, however, this u_k does not belong to the original U. Theorem 4 gives the basis for our algorithm. All we have to do is to generate the semigroups UGS and VGS using the UG-s and VG-s generators. For meet idempotency both UGS and VGS are finite.

5. THE ALGORITHMS

ALGORITHM 1

Meet forming:

First step

Select row 1 in the R_ϕ table, i.e. consider the element u_1. Form

$$\phi(u_1)\cap\phi(u_i) \quad \text{for all} \quad i>1,\ i\in I \quad .$$

Second step

Decide whether there is a row being equal to one of the meets have already been formed, i.e. decide whether there exists a $u_k \in U$ such that for some $u_i \in U$

$$\phi(u_1)\cap\phi(u_i) = \phi(u_k)$$

If not, introduce u_{m+1} for the first /smallest/ is such that $\phi(u_1)\cap\phi(u_i)$ is not occuring in the R_ϕ table as a row. Make up a table with $u_1, u_2, \dots, u_\ell$ ... as both row - and column - headings and fill in the result of second step. In the other case, into the meet of the u_1 row and the u_i-th column put k. This table will be called U-Meet table /UM-table/.

Example:

According to Table I /which is an R_ϕ-table/ we have

$$m = 18, \quad n = 7 \ .$$

Here for instance:

$$\phi(u_1) = \{v_1\}$$

and

$$\phi(u_2) = \{v_1, v_5, v_7\} \quad .$$

In this case

$$\phi(u_1) \cap \phi(u_2) = \{v_1\} = \phi(u_1)$$

Third step

Repeat the procedure in step two for u_2, u_3 ..., u_m,..., u_n until meeting yields no new element.

Extend the R_ϕ-table in "U-direction", i.e. if for both u_{j1}, and u_{j2}

$$u_{j1}\phi v_i \qquad j_1 \leq \bar{m} \; , \; i \leq n$$

and

$$u_{j2}\phi v_i \qquad j_2 \leq m$$

then stipulate that

$$u_{m+k}\phi v_i \quad .$$

Example: See Table I. We have

$$\phi(u_7) \cap \phi(u_{11}) = \phi(u_{27}) = \phi(u_{18+9}) \; .$$

For both u_7 and u_{11} we have $u_7\phi v_6$ and $u_{11}\phi v_6$, therefore we stipulate that $u_{27}\phi v_6$.

Accordingly, a cross is put into the cell belonging to the 27^{th} row and 6^{th} column in our U-extended R_ϕ-table. See Table II.

In our R_ϕ-table /Table I/ m=18, and /II/ contains 32 nonzero elements /$\bar{m}$=33/. Zero element /u_8/ is a V-generator. Our UM-table can be seen in Table III.

Fourth step

Carry the procedure, described in step two and three, for generators v_1 $v_2,\dots,v_n$ out, but take into consideration that sets $\phi^+(v_j)$, in general, may contain u_i with $i>m$. In other words forming the VGS semigroup use the U-extended R_ϕ-table. This way one gets the V-extended R_ϕ-table.

Fifth step

Form the V meet table /VM-table/.

ALGORITHM 2

Establishing dual isomorphism between the semigroups factorization

Let u_i /$i\leq m$/ be an arbitrary element of UGS. Using UM-table one can "factor" it, i.e. producing in a form of meet /or product/ of generator elements i.e. with index $i\leq m$. The algorithm goes as follows:

First step

Check whether all the U-generators are independent, i.e. whether they are not meets of each other. In other words factorise even the generators, too. Select an arbitrary element u_i /$i\leq m$/. Enter the i-th column of the UM-table. Select all the rows /with index not greater than m/ having i in column i. The headings of these rows will be the factors of u_i.

Example /see Table IV/
Consider u_{19}. Entering the 19-th column of the UM-table /Table III/ we find that

row No. 6,
row No. 7,
row No. 9,
row No. 15,
and row No. 18

has 19 in the 19-the column. So the factors of u_{19} are just u_6, u_7, u_9, u_{15} and u_{18} and there is no other factor. So we have

$$\phi(u_{19}) = \phi(u_6) \cap \phi(u_7) \cap \phi(u_9) \cap \phi(u_{15}) \cap \phi(u_{18}) \; .$$

In this case all the factors are <u>prime</u> /having no factor different from itself and the unity i.e. u_{18}/. In general, however, not every V-generator is prime. E.g. u_{12} is a V-generator /being 12<18/, but not prime for,

$$\phi(u_{12}) = \phi(u_2) \cap \phi(u_6) \cap \phi(u_{14}) \cap \phi(u_{17}) \; .$$

So, if necessary, u_{12} could have been omitted at the outset.

<u>Second step</u>

Matching the UF-table and the U-extended R_ϕ-table make up the dual isomorphism-table /φ-table/. Matching is carried out on the basis that for each $u_i \in$ <u>UGS</u> /$i \leq \bar{m}$/ and for each $v_k \in$ <u>VGS</u> /$k \leq \bar{m}$/ we can establish the following equalities simultaneously

$$\phi(u_i) = v_{j1} \cup v_{j2} \cup \ldots = \phi(u_{i1}) \cap \phi(u_{i2}) \cap \ldots =$$

$$= u_{i1} \cap u_{i2} \cap \ldots \; ,$$

$$\phi^+(v_k) = u_{11} \cup u_{12} \cup \ldots = \phi^+(v_{\ell 1}) \cap \phi^+(v_{\ell 2}) \cap \ldots =$$

$$= v_{k1} \cap v_{k2} \cap \ldots$$

Here, according to the theory, lattice theoretic join operation, $\cup$ is meant by

$$u_i \cup u_k = \varphi(\Phi(u_i) \cup \Phi(u_k))$$

$$i,j,k,\ell \leq \bar{m}$$

$$v_j \cup v_\ell = \varphi^+(\phi^+(v_j) \cup \phi^+(v_\ell))$$

Now, on the other hand, expressions $v_{j1} \cup v_{j2} \cup \ldots$ and $u_{i1} \cup u_{i2} \cup \ldots$ are easily recognized for

$$v_{j1} \cup v_{j2} \cup \ldots = \varphi(\{v_{j1}\} \cup \{v_{j2}\} \cup \ldots) = \varphi(v_{j1}, v_{j2}, \ldots) =$$

$$\text{/being a closed set/} \doteq \{v_{j1}, v_{j2}, \ldots\} \quad .$$

Now, this set is immediately given by the U-V extended R_ϕ-table. Matching itself consists of pairing sets with

$$\{j1,\ j2,\ \ldots\} = \{\ell 1,\ \ell 2,\ \ldots\} \quad .$$

<u>Example</u>

Consider u_{20}. <u>First</u> from the UF-table /Table IV/ we see that

$$u_{20} = u_2 \cap u_6 \cap u_{14} \cap u_{17} \cap u_{18} \quad .$$

Secondly, on the other hand, u_{20} <u>as a set</u> contains two elements viz. v_1 and v_5 i.e.

$$\phi(u_{20}) = \{v_1, v_5\} = \{v_1\} \cup \{v_5\} \quad .$$

Thirdly, from the VF-table /not shown here/, we have:

$$v_1 \cap v_5 = v_{11}$$

so we infere:

$$\phi(u_{20}) = v_{11}$$

or equivalently

$$\phi^{+}(v_{11}) = v_{20}$$

Finally, from the U-V-extended R_{ϕ}-table, /not shown here/ however, we have

$$v_{11} = \{u_2, u_6, u_{14}, u_{17}, u_{18}\} \; .$$

This way we have made up our φ-and φ^{+}-tables. See Table V.

As a byproduct we have established all the subalgebras of UGS and VGS /moreover even that of the lattices L, $\underline{L}^{+}$/. The reason is simply that an element of UGS /VGS/ as a set is a subalgebra of UGS /VGS/. E.g. the element v_{11} as the set

$$\{u_2, u_6, u_{14}, u_{17}, u_{18}\}$$

means a set which is closed under the semigroup operation $\cap$.

In possession of all the tables we have produced the diagrams of both, the lattices $\underline{L}$ and $\underline{L}^{+}$ can be drawn. Actually a telescopized version of the diagrams of L and L^{+} is on Figure 1, but it should be noted that for drawing we have given no algorithm. By the way using φ and φ^{+} tables, it is quite immediate to construct lattices L and L^{+}.

TABLE I

R_ϕ-table for a binary relation with m=18, n=7

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
u_0								
u_1		+						
u_2		+				+		+
u_3							+	+
u_4								+
u_5					+			
u_6		+	+			+		
u_7		+	+	+			+	
u_8								
u_9		+	+				+	
u_{10}				+			+	+
u_{11}			+	+	+		+	+
u_{12}						+		
u_{13}					+	+		
u_{14}		+		+	+	+		+
u_{15}		+	+	+	+		+	+
u_{16}		+			+			
u_{17}		+		+	+	+		
u_{18}		+	+	+	+	+	+	+

TABLE II

U-extended R_{ϕ}-table with m=18, $\bar{m}$=33

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
u_1	+						
u_2	+				+		
u_3						+	+
u_4							+
u_5				+			
u_6	+	+			+		
u_7	+	+	+			+	
u_8							
u_9	+	+				+	
u_{10}			+			+	+
u_{11}		+	+	+		+	+
u_{12}					+		
u_{13}				+	+		
u_{14}	+		+	+	+		+
u_{15}	+	+	+	+		+	+
u_{16}	+			+			
u_{17}	+		+	+	+		
u_{18}	+	+	+	+	+	+	+
u_{19}	+	+					
u_{20}	+				+		
u_{21}			+	+			
u_{22}			+				
u_{23}						+	
u_{24}		+				+	
u_{25}	+		+				
u_{26}	+						+
u_{27}		+	+			+	
u_{28}		+				+	

Table IV

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
u_{29}			+			+	
u_{30}			+				+
u_{31}			+	+			+
u_{32}	+		+	+			
u_{33}	+		+	+			+

TABLE III

UM-table /U-meet table/ /just par of it/

ui	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	24...33
1																						
2											12											
3																						
4																						
5																						
6											12								19			
7										27									19			
8																						
9																						
10																			19			
11																						
12																						
13																						
14											12											
15																			19			
16																						
17											12											
18											12								19			
19																						
20																						
21																						
22																						
23																						
24																						
25																						
26																						
27																						
28																						
29																						
30																						
31																						
32																						
33																						

TABLE IV

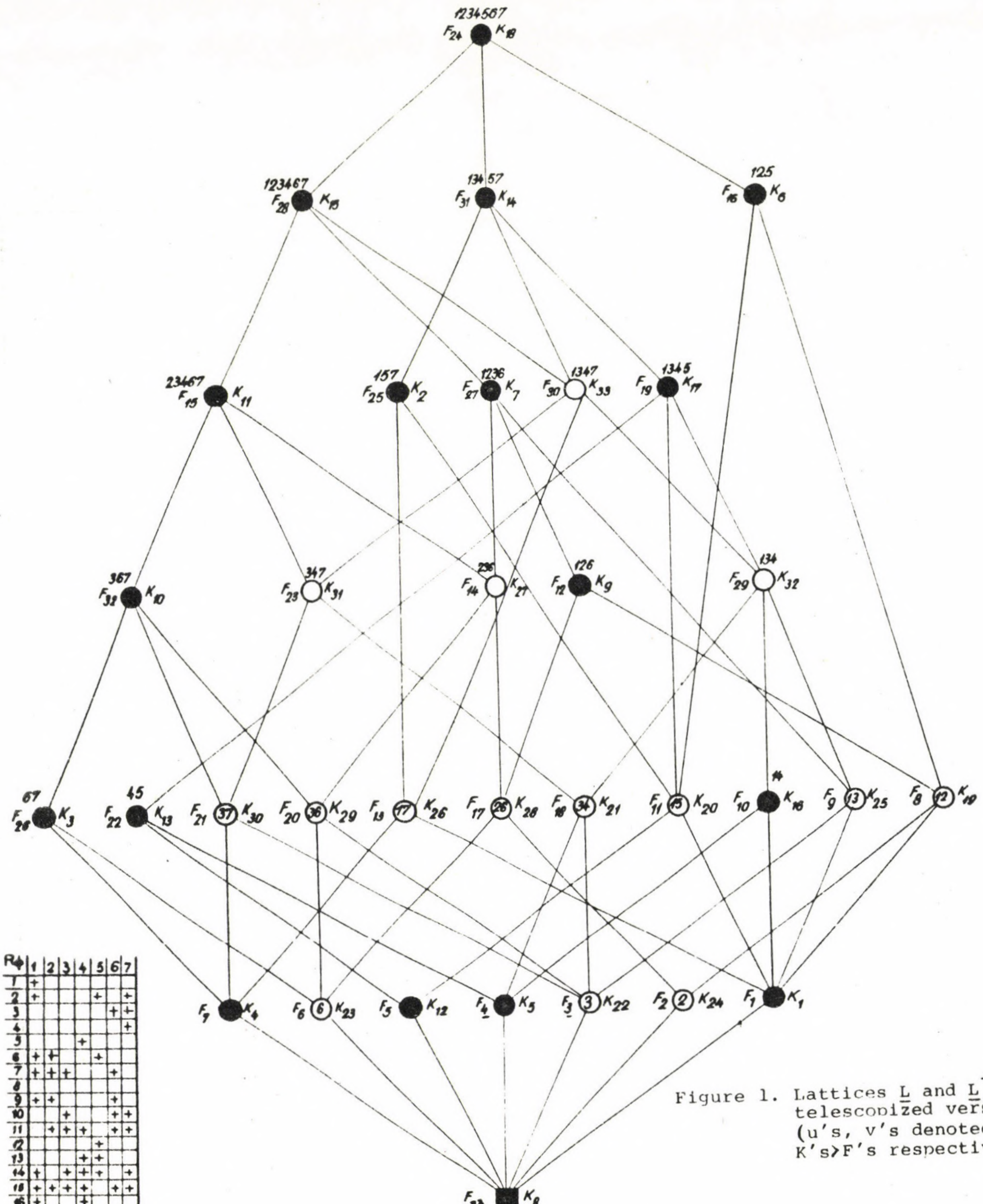
UF-table /U-factor table/ /just a fragment/

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}
u_1	+	+																
u_2																		
u_3																		
u_4																		
u_5																		
u_6																		
u_7																		
u_8																		
u_9																		
u_{10}																		
u_{11}																		
u_{12}	+					+								+		+		+
u_{13}																		
u_{14}																		
u_{15}																		
u_{16}																		
u_{17}																		
u_{18}																		
u_{19}						+	+		+						+			+
u_{20}	+					+											+	+
u_{21}																		
u_{22}																		
u_{23}																		
u_{24}																		
u_{25}																		
u_{26}																		
u_{27}																		
u_{28}																		
u_{29}																		
u_{30}																		
u_{31}																		
u_{32}																		
u_{33}																		

TABLE V

/fragment/

φ-table X	/φ-closed sets $X \subset U$/ /image of X/ $\phi(X)$
$\{u_1, u_2\}$	$\{v_6, v_7, v_9, v_{15}, v_{18}\}$
$\{u_1, u_2, u_3, u_4, u_6, u_7\}$	$\{v_{15}\}$
$\{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$	$\{v_{18}\}$
$\{u_1, u_2, u_3, u_6\}$	$\{v_7\}$
$\{u_1, u_2, u_5\}$	$\{v_6\}$
$\{u_1, u_2, u_6\}$	$\{v_9\}$
$\{u_1, u_3\}$	$\{v_7, v_{14}, v_{15}, v_{17}, v_{18}\}$
$\{u_1, u_3, u_4\}$	$\{v_{14}, v_{15}, v_{17}, v_{18}\}$
$\{u_1, u_3, u_4, u_5\}$	$\{v_{17}\}$
$\{u_1, u_3, u_4, u_5, u_7\}$	$\{v_{14}\}$
$\{u_1, u_3, u_4, u_7\}$	$\{v_{14}, v_{15}, v_{18}\}$
$\{u_1, u_4\}$	$\{v_{16}\}$
$\{u_1, u_5\}$	$\{v_1, v_6, v_{14}, v_{17}, v_{18}\}$
$\{u_1, u_5, u_7\}$	$\{v_2\}$
$\{u_1, u_7\}$	$\{v_1, v_{14}, v_{15}, v_{18}\}$



R	1	2	3	4	5	6	7
1	+						
2	+				+		+
3						+	+
4							+
5				+			
6	+	+			+		
7	+	+	+			+	
8							
9	+	+				+	
10			+			+	+
11		+	+	+		+	+
12					+		
13				+	+		
14	+		+	+	+		+
15	+	+	+	+		+	+
16	+			+			
17	+		+	+	+		
18	+	+	+	+	+	+	+

Figure 1. Lattices $\underline{L}$ and $\underline{L}^+$ in a telescopized version (u's, v's denoted by K's>F's respectively)

REFERENCES

[1] Bar-Hillel, Y. /1952/, An Outline of a Theory of Semantic Information. In Bar-Hillel /1964/ pp. 221-274

[2] Bar-Hillel, Y. /1964/, Language and Information. Addison Wesley 1964.

[3] Carnap, R. /1942/, Introduction to semantics, Harvard University press.

[4] Chomsky, N. and M.P. Schützenberger:/1961/, The algebraic theory of context-free languages. In: Computer programming and formal systems. Studies in logic and the foundations of mathematics. /P.Bradford and D.Hirschberg eds./ North Holland, Amsterdam, 1963.

[5] Codd, E.F. /1970/, A Relational Model of Data for Large Shared Data Banks. Comm. ACM 13, 377-384.

[6] Farradane, J.E.L. /1966/, Report on Information Research Centre for Information Science, The City University, London.

[7] Fay, G. /1970/, Phenomenological foundation of quantum logic. Acta Physica Hungarica, 29, 27-33.

MECHANICAL ANALYSIS OF HUNGARIAN WORD FORMS

György HELL
Technical University, Budapest
Institute of Languages
Budapest, Hungary

1. PRELIMINARY NOTES

Work on mechanical analysis of natural languages began in Hungary in the early sixties with the aim of obtaining an algorithm for machine translation. This activity resulted in a number of papers on ways of formal analysis in the field of Russian morphology and syntax and on synthesis of Hungarian word forms.

With the end of the "translation era" formal analysis has been extended to statistical investigations /datas on vowel and consonant frequency, frequency of consonant clusters, syllable structures, word length etc./, mechanical syllabification and vocabulary studies. /A characteristic result of this period was the edition of the Hungarian a-tergo vocabulary./

In the last years interest has turned towards the possibility of analysis in Hungarian with the principal aim of syntactic parsing. The first step towards this goal was the construction of a working word form analyser. Some details of this work are given in the following chapters.

2. FINITE STATE GRAMMAR FOR WORD FORM GENERATION

We may have enough assurance for the feasibility of a morphological analysis of Hungarian word forms if we succeed in building an algorithm which is capable to generate Hungarian word forms and to check them in a sense, that uncorrect words are identified as such and not accepted for analysis.

According to the grammatical description, Hungarian word forms are composed of stems t , word forming suffixes k , case endings r , verbal prefixes and plural endings. Characteristically for an agglutinative language, all these elements can occur in different combinations, e.g.:

nyelv + tan + oktat + ás + ban
t + t + t + k + r
/= in grammar teaching/

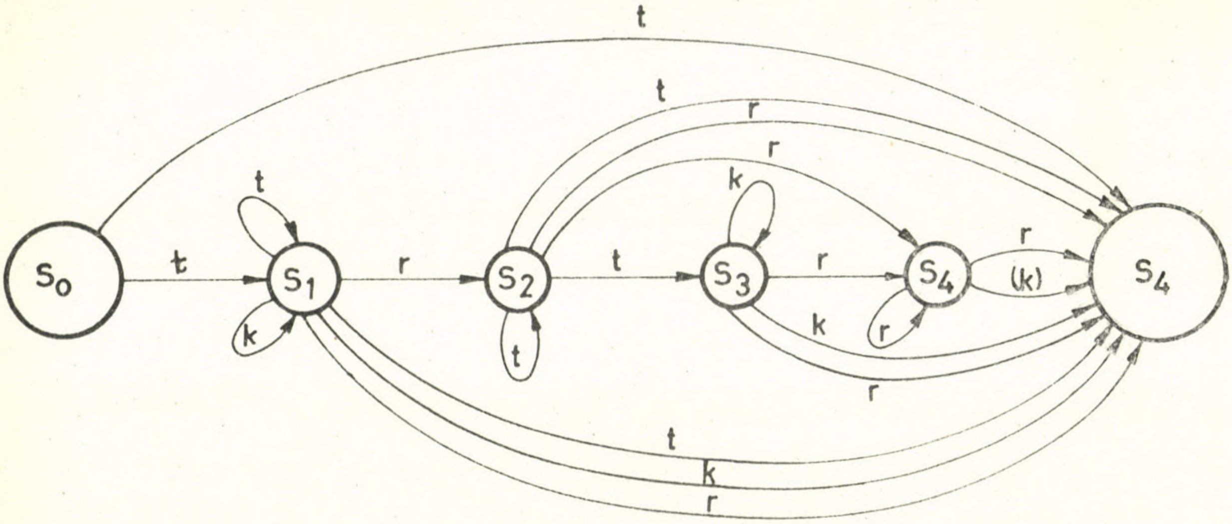
nyelv + tan + könyv + ei + nk + ben
t + t + t + r + r + r
/= in our grammar books/

távol + ba + lát + ás
t + r + t + k
/= television/

To make the generation and control process easier, we reduced the five components to three, by taking the plural endings equal to case endings and the verbal prefixes as belonging to the verb stems. Such a simplification brings no significant differences into the accepted classification. Another presumption requires that some endings should be taken to the word stems and forms as nekem /to me/, tőled /from you/, hozzátok /towards you/ should be considered as consisting of stem + ending. /Such a presumption is not absolutely necessary because the words nekem, tőled etc.

can be accepted as elliptic forms out of énnekem, tetõled, tihozzátok in which case we have the structure: /stem/ ending, ending./

With the restrictions given above, Hungarian word forms can be generated by a finite state grammar:



Here S_o, S_1, S_2, S_3, S_4 and S_v give the states of the grammar /or automaton/ with S_o as the initial and S_v the final state. The arrows between the states give the possibilities of transition from one state to the other, the letters on the arrows signify the morpheme-types obtained by transition.

In this system several paths go from one state to the next and so the grammar belongs to the indefinite finite state grammar type.

The transcription rules for generating Hungarian word forms can be obtained from the diagram.

Nos.of rules	
1.	$S_o \rightarrow t\ S_1$
2.	$S_o \rightarrow t\ S_v$
3.	$S_1 \rightarrow t\ S_1$
4.	$S_1 \rightarrow t\ S_v$
5.	$S_1 \rightarrow k\ S_1$

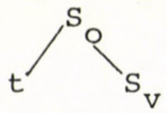
Nos.of rules	
6.	$S_1 \to k\ S_v$
7.	$S_1 \to r\ S_2$
8.	$S_1 \to r\ S_v$
9.	$S_2 \to t\ S_2$
10.	$S_2 \to t\ S_3$
11.	$S_2 \to t\ S_v$
12.	$S_2 \to r\ S_4$
13.	$S_2 \to r\ S_v$
14.	$S_3 \to k\ S_3$
15.	$S_3 \to r\ S_4$
16.	$S_3 \to k\ S_v$
17.	$S_3 \to r\ S_v$
18.	$S_4 \to r\ S_4$
19.	$S_4 \to r\ S_v$

This system of rules where capital letters represent categorial symbols and lower case letters stand for terminal symbols /in our case: morpheme types/, allows to produce all the Hungarian word forms, even such as igénybevétel /utilization, making use of/, karbantartás /maintenance/ etc., i.e. words having two stems with a case ending between them. Words as nagybani /as on a large scale, in gross/ containing a word building suffix after a case ending could be produced by the finite state grammar if state S_4 is connected with state S_v. /In our diagram this is shown by a dotted line./

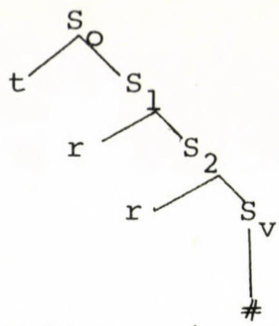
For producing the word end we complete our rule system with a 20th rule:

20. $S_v \longrightarrow \#$

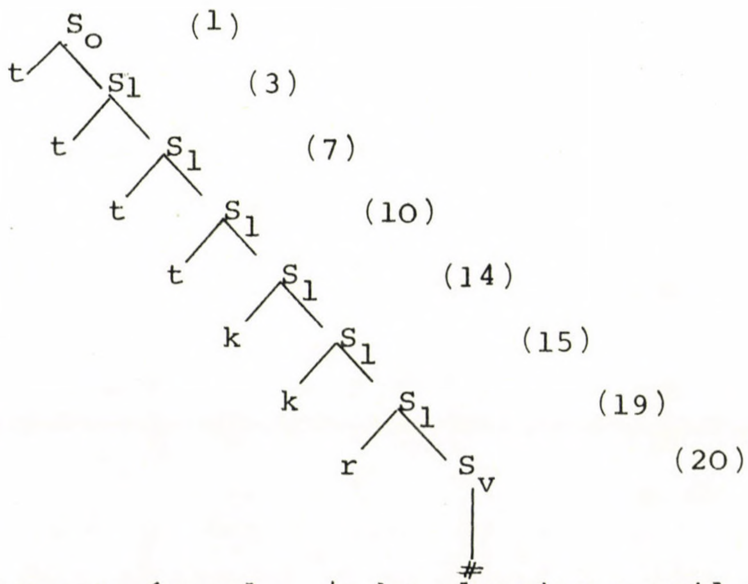
Let us see some examples, how word forms are produced.



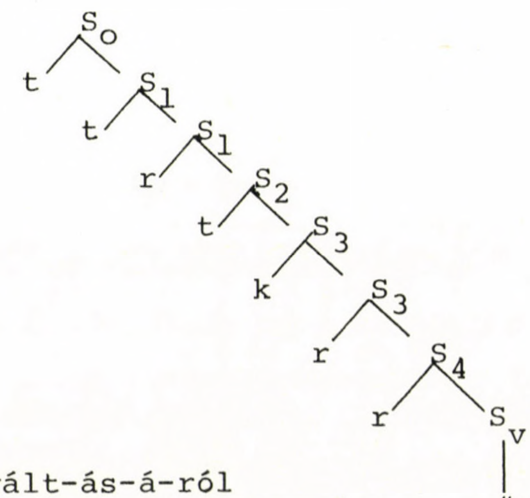
ház /= house/



ház-am-ban /= in my house/
/house/my/in/



vas-út-vonal-terv-ez-és-sel /= by planning a rail road line/
/iron-road-line-plan-ing - by/



apró-pénz-re-vált-ás-á-ról /=about changing
/small-coin-to-change-ing-it-about/ # it to small coins/

There is, however, no restriction in the rules, which would define how many word stems, word building suffixes or case endings can follow each other in a correct Hungarian word form, but no Hungarian grammar gives an exact definition of this problem.

If we want to use the grammar for checking a given word whether it is built according to the Hungarian word constructi g rules, our generating rules have to be transformed: the automaton must be given the "input signals" representing the components of the given word, furthermore the states which accept the input, and as an output a new state for receiving the next component.

The new rules are obtained by transformation out of rules II.

Nos.	input	states
I	$t \rightarrow S_o$	S_1
II	$t \rightarrow S_o$	S_v
III	$t \rightarrow S_1$	S_1
IV	$t \rightarrow S_1$	S_v
V	$k \rightarrow S_1$	S_1
VI	$k \rightarrow S_1$	S_v
VII	$r \rightarrow S_1$	S_2
VIII	$r \rightarrow S_1$	S_v
IX	$t \rightarrow S_2$	S_2
X	$t \rightarrow S_2$	S_3
XI	$t \rightarrow S_2$	S_v
XII	$r \rightarrow S_2$	S_4
XIII	$r \rightarrow S_2$	S_v
XIV	$k \rightarrow S_3$	S_3
XV	$r \rightarrow S_3$	S_4

Nos.	input	states
XVI	$k \rightarrow S_3$	S_v
XVII	$r \rightarrow S_3$	S_v
XVIII	$r \rightarrow S_4$	S_4
XIX	$r \rightarrow S_4$	S_v
XX	$\# \rightarrow S_v$	S_o

The demonstrate how these rules work, let us take two morpheme conbinations. One of them corresponds to a Hungarian word form, the other does not.

t,t,k,k,r,r,# = érdekházasságokról /about marriages of convenience/

*t,r,r,t,#

The automaton checks the forms in the following way

I	/t, S_o, S_1/ t, k, k, r, r,#
III	t, /t, S_1, S_1/ k, k, r, r,#
V	t, t, /k, S_1, S_1/ k, r, r,#
V	t, t, k, /k, S_1, S_1/ r, r,#
VII	t, t, k, k, /r, S_1, S_2/ r,#
XIII	t, t, k, k, r, /r, S_2, S_v/,#
XIX	t, t, k, k, r, r, /#, S_v, S_o/

On the left side the rule numbers are given, the exact form of the rule is in parenthesis inside the word form after the morpheme type scanned by the rule. If the rules can proceed through the sequence of morphemes, the combination is accepted as a genuine Hungarian word form.

Sequence 2.

I	$/t,\ S_o,\ S_1/\ r,\ r,\ t,\ \#$
VII	$t,\ /r,\ S_1,\ S_2/\ r,\ t,\ \#$
XII	$t,\ r,\ /r,\ S_2,\ S_4/\ t,\ \#$
	$t,\ r,\ r,\ /t,\ -\ -/\ \#$

The process stops at the fourth step, even if rule XIII is taken instead of rule XII.

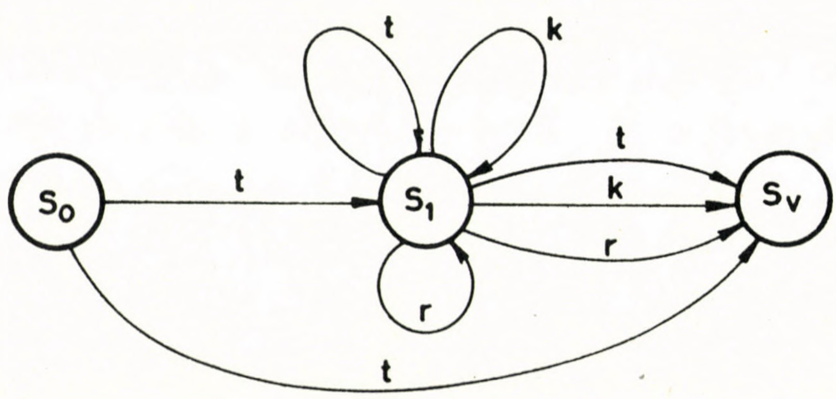
PRACTICAL ANALYSIS OF WORD FORMS

A segmentation of Hungarian word forms on a computer differs in some respects by a theoretical analysis. In the elaboration of the rules of analysis we took it for granted that word forms are given as morpheme constructions and the analysis was carried out on a string of morphemes. In real analysis word forms are given as concatenations of letters and nothing is known about the structure of the word. The morpheme structure of a word can be obtained only if we can identify some successive parts of it as elements of different morpheme lists representing stems, case endings, word building suffixes, and the sequence of the different morpheme types in the word form corresponding to a possible Hungarian word structure.

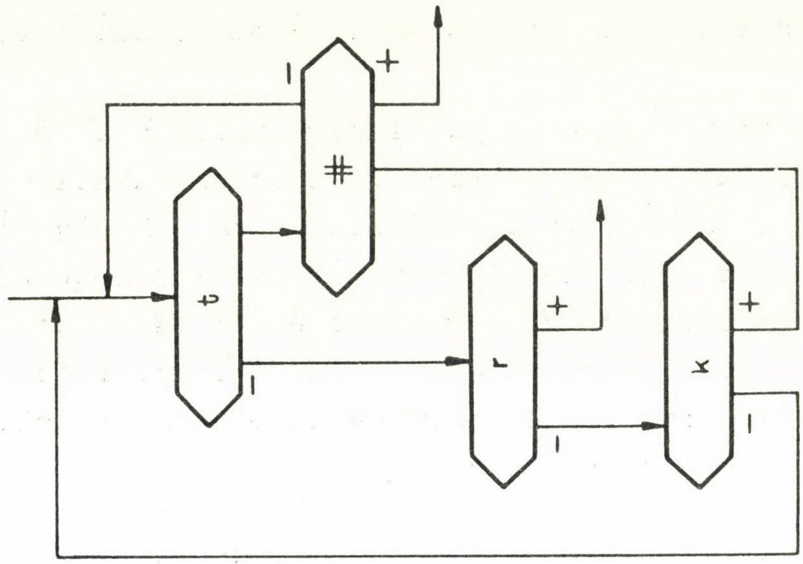
In the identification process following difficulties may arise: some words contain letter sequences identical with stems or endings but what they are not in the given word. E.g. <u>víg</u> /=merry/ and <u>asztal</u> /=desk/ in <u>vígasztal</u> /to console/, other words can be analyzed as compounds or suffixated forms: <u>karóra</u> /=wrist watch/onto a post/ and still other morphemes or morpheme combinations represent different morpheme types: <u>ének</u> /=to his or her sthg/song/, <u>ikre</u> /=onto their sthg/one of his or her twins/, okkal /=with your sthg/with reason/ etc. All such cases occur in Hungarian more often than in other European languages because of its agglutinative character.

In a word analysis we can only partially overcome such difficulties. We can, however, require that in identifying possible morpheme components in the word form our algorithm should always accept only the longest identified sequence of letters, but this strategy does not solve all cases of possible ambiguity. In some cases it leads moreover to uncorrect results which can be eliminated on   by a following syntactical or semantic analysis.

There is also a second point, why a practical analysis differs from a theoretical one. Analysing Hungarian texts we can soundly suppose that all the words have a correct construction and so a checking on correctness is not necessary. Hence our analysing algorithm becomes much simpler than in its theoretical form:



The block diagram for obtaining the longest possible morpheme component in the word has the following form:



LITERATURE

[1] Dömölki,B.: An Algorithm for Syntactic Analysis. CL and CL III. 29-46 pp.,1964.

[2] Kelemen,J.: Über die Experimente an einem sprachstatistischen Automaten, CL and CL V. 149-157 pp.,1966.

[3] Klauszer,J.: Some Problems of Synthesis of the Hungarian Noun Forms, CL and CL II. 111-126 pp.,1964.

[4] Sipőczy,Gy.: Some Semantic Aspects of the Machine Translation from Russian into Hungarian, CL and CL II. 159-178 pp., 1964.

[5] Reverse-Alphabetized Dictionary of the Hungarian Language, compiled by Papp Ferenc, Akadémiai Kiadó, Budapest, 1969.

[6] Ботош, И.: Опыт автоматического анализа текстов на языке эсперанто, СЛ и СЛ V. 19-40, 1966.

[7] Хелл, Дь.: Определение номинальных групп в МП с русского на венгерский, СЛ и СЛ I. 5-107, 1963.

FINITE GEOMETRICAL DATA BANK BY GALOIS ALGORITHM*

G. FAY and Mrs D.V. TAKÁCS
Institute for Economy Organisation and
Computational of Metallurgy and Engineering Industry

INTRODUCTION

Bose et al /1967/ have shown how finite geometries can be applied in constructing data banks. /File Organization Schemes./ Actually their constructions need solutions of equation systems in Galois fields which seems to be quite difficult to implement. Sets being candidates for the elements /i.e. lines, planes, hyperplanes, etc./ of a finite /projective/ geometry turn, however, ought to be the closed sets of a suitably defined binary relationship between the points of this geometry. It is true, on the other hand that this class of geometries is rather narrow viz. those above GF $/2^n/$. Fay /1973/ has developed a technique yielding an algorithmic production of a Galois-connection between two given finite sets U and V with respect to a relation ϕ /CUxV/. We shall call this algorithm "Galois-algorithm", while a "Galois-connection" /which is not generally defined as such/ is meant the one-one correspondence between the closed subsets of U and V. "Closed" here means closed with respect to a closure operation "induced" by ϕ. For preliminaries see Szász /1963/ and Fay /1973/. Galois algorithm, by the way, involves no need for solution of equations whatsoever. Notwithstanding, it will not be used here in a straightforward

* This work has been supported by the Hungarian Ministry for Metallurgy and Machine Industry under Contract No Y-12.172 /68 Cp.68.9/1.

manner, rather, upon its formal characteristics a still simpler form of algorithm is developed for producing /certain/ finite geometries. This algorithm is immediate using the technique to find out all the closed "boolean subspaces" of a set. "Closed" here, in turn, means closeness under a boolean ringsum operation.

Needless to say that data banks are in the strongest interactions with artificial languages. In a sense Boses' approach to data banks can - in our opinion - be considered as a sort of a geometrical language approach in which all the places for the data to come are selected out apriori. This selection is highly algorithmic and can indeed be very effective with respect to data handling.

Also, it can be considered as a "coordinatization" of the data space. The "buckets", as the selected boxes for data are called, are, on the other hand, the conveyors of certain relations /as collections of attributes/. Therefore the finite geometrical approach of data banks has something to do with the relational data bank systems. This latter branch of investigations into data bases has seemingly been developed quite independently at IBM San Jose by a group from 1967 centering around E.F.Codd /1967/.

In both aspects certain algebraic operations can be performed upon relations representing collections of data. In the finite geometrical management /as we have shown in this paper/ these operations are lattice theoretical ones /finite geometries being lattices/, whereas in the relational file organization systems these are other algebraic but probably again lattice theoretical operations /such as projection or join of relations/.

It is felt that some light can be thrown to the connection between these two ways of file organization /or data bank construction/ by observing that in both ways the problem of the so-called "conceptual processing of data" is of vital importance. The user is not satisfied by possessing all the records pertaining to a query. He wants naturally more than this. He wants to get an overview of the data, to discover their factual structure, to uderstand data rather than barely having them.

But how to "conceptualize" data? In the literature there cannot be found anything like "conceptual data processing" although it takes place every minute within our brains.

We suggest /and tried to support in another article of Fay (1973)/ that a set of objects /records etc./ can be considered as a representative of a concept with respect to a given system - frame - of attributes, properties, features if /and only if/ the set is closed under a Galois connection /between objects ond attributes/. We show here actually that the buckets in Bose's finite geometrical data bank are indeed closed subspaces under a suitable closure therefore from a "conceptual" point of view they can be considered as representing notions. As for the relational data banks we will try to show, in a next paper, how it is embeddable into a bit more general technique by which both, the operations /between relations/ and the "notionlike" sets can /algorithmically/ be produced. This "more general technique" will turn out to be the good old edge notched card technique in a somewhat obstruse form so as to be implementable in a suitable electronical medium.

/As for the medium - by the way - we envisage a cellular automaton.

1. BASIC CONCEPTS

Throughout this paper the following concepts and notations are accepted. As for the details see Fay /1973/ and Szász /1963/. The binary relation Φ_n between the elements of a set

$$U_n = \{u_o, u, \ldots, u_{2^n-1}\},$$

the R_Φ-table that relation /denoted by $R\,\Phi_n$/; the closure operation φ_n

$$\varphi_n(X) = \Phi_n(\Phi_n(X)), \quad X \subset U_n, \; \Phi_n(X) = \bigcap_{x \in X} \Phi_n(X)$$

$$\Phi_n(X) = \{y \mid x \, \Phi_n \, y\}.$$

A set $X \subset U_n$ is called φ-closed iff $\varphi_n(X) = (X)$

Closure φ_n is said to be induced by the relation Φ_n.

U-generators are just sets of form $\Phi_n(u)$, $u \in U_n$, the table of the relation Φ_n /or similarly of a ψ_n/ is denoted by $R\,\Phi_n$ /or $R\,\psi_n$/.

2. U-GENERATORS AND BOOLEAN SPACES

Beginning with the set

$$U_n = \{u_o, u_1, u_2, \ldots, u_{2^n-1}\}, \qquad n=1,2,\ldots$$

let us define a relation $\Phi_n \subset U_n \times U_n$ between the elements of U_n. The definition is recursive, and in this concise form is due to G.T.Herman /1973/.

DEFINITION /of $R\Phi_n$/

$$R\,\Phi_1 = \begin{array}{|c|c|}\hline + & \\ \hline & \\ \hline\end{array}$$

$$R\,\Phi_n = \begin{array}{|l|l|}\hline R\,\Phi_{n-1} & R\,\Phi_{n-1} \\ \hline R\,\Phi_{n-1} & R\,\psi_{n-1} \\ \hline\end{array}$$

where

$$R\,\psi_n = \begin{array}{|l|l|}\hline R\,\psi_{n-1} & R\,\psi_{n-1} \\ \hline R\,\psi_{n-1} & R\,\Phi_{n-1} \\ \hline\end{array} \qquad \text{with} \quad R\,\psi_1 = \begin{array}{|c|c|}\hline & \\ \hline & + \\ \hline\end{array}$$

Figure 1. shows the $R\,\Phi_n$ table for $n=4$.

There is an easy consequence of this definition:

LEMMA 1.

Let $\quad k \in \{0, 1, 2, \ldots, 2^{n+1}\}$, and $u_i, u_j \in U_{n+1}$,

Let

$$k^{*} = \begin{cases} k & \text{if } \ k < 2^n \\ k-2^n & \text{if } \ k \geq 2^n \end{cases}$$

Then

$$u_{i^*}\ \Phi_n\ u_{j^*} \quad \text{iff} \quad u_i\ \Phi_{n+1}\ u_j$$

Proof: Immediate.

Having defined relation Φ_n ($\subset U_n \times U_n$) we can speak of the set $\Phi_n(X)$ for any set $X \subset U_n$ especially of U-generators. Owing to the special features of Φ_n a deeper insight into the algebraic structure of the U-generators may be obtained.

DEFINITION

Let u_i, $u_j \in U_n$ and resolute i and j into binary digits:

$$(i,j \in \{0,1,\dots,2^n-1\})$$

$$i = \sum_{k=1}^{n} i_k^{2-k} \; , \; j = \sum_{\ell=1}^{n} j_\ell^{2^{n-\ell}} \; , \; i_k, \; j_\ell \in \{0,1\}$$

Meaning ring sum in $\{0,1\}$ as usual /i.e. $0 \oplus 0 = 1 \oplus 1 = 0$, $0 \oplus 1 = 1 \oplus 0 = 1$/

we define:

$$i \oplus j = \sum_{k=1}^{n} (i_k \oplus j_k)2^{n-k}$$

Finally let, by definition, $u_i \oplus u_j = u_{i \oplus j}$.
The /unique/ zero element of this operation will be denoted by 0 or u_0 alternatively /as convenient/. Sometimes we write i instead of u_i /especially in table headings/ unless misunderstanding occurs. Similarly, $u_i \le u_j$ means $i \le j$, or e.g. $u_i - 2^n$ means u_{i-2^n}. Properties of ring sum are well-known. See e.g. Szász /1963/ pages 126-130. Out of them we mention only these:

LEMMA 2.

For any u_1, $u_j, u_k \in U_n$ the following equations are equivalent:

$$u_j \oplus u_j \oplus u_k = 0, \quad u_j \oplus u_j = u_k, \quad u_j \oplus u_k = u_i, \quad u_k \oplus u_j = u_j \; .$$

The following concept of "boolean space" intends to overcome the difficulties arising in finite vector spaces.

DEFINITION

A set $S \subset U_n$ is called a <u>boolean space</u> if

$$u_i \; , \; u_j \in S \text{ implies } u_i \oplus u_j \in S.$$

Boolean spaces have a number of simple properties out of which a few /needed below/ are listed in lemmas 3.-5.

LEMMA 3.

Every boolean space contains /the/ zero element.

Proof: Let $S=\{s_o,s_1,\dots,s_k\}$. If zero were not contained in S then for any $s_i \in S$ $O=s_j \oplus s_i \subset S$ would be a contradiction.

LEMMA 4.

The intersection of two boolean spaces is a boolean space again.

Proof: Let S, T be BS's /boolean spaces/ and let $u_i, u_j \in S,T$. Then $u_i \in S$, $u_j \in S$, and $u_j \in T$, $u_i \in T$. But S, T being BS's it follows $u_i \oplus u_j \in S$, T implying $u_i \oplus u_j \in S \cap T$.

LEMMA 5.

Every U-generator is a boolean space.

Proof: Suppose, inductively, that for any $u \in U_n$ with a fixed n the U-generator

$$\Phi_n(u) = \{v | u \; \Phi_n \; v\}$$

<u>is</u> a boolean space. Furthermore, suppose that for some u_i, u_j, $u_k \in U_{n+1}$ we have

$$u_i \; \Phi_{n+1} \; u_k \quad \text{and} \quad u_j \; \Phi_{n+1} \; u_k \tag{1}$$

Making use of Lemma 2, without loss of generality, we may assume that

$$u_i \leq u_j \leq u_i \oplus u_k \quad .$$

Of course, if $u_i \; u_j \; u_k \in U_n \subset U_{n+1}$ there is nothing to prove. If, in turn $i,j,k \geq 2^n$ then on one hand:

$$i^{\mathbf{x}},\; j^{\mathbf{x}},\; k^{\mathbf{x}} \leq 2^n$$

On the other hand, by Lemma 1,

$$u_i \; \Phi_{n+1} \; u_k \quad \text{implies} \quad u_{i^*} \; \Phi_n \; u_{k^*} \; ,$$

$$u_i \; \Phi_{n+1} \; u_k \quad \text{implies} \quad u_{j^*} \; \Phi_n \; u_{k^*} \; .$$

So, by the inductive assumption from $u_{i^*} \; \Phi_n \; u_{k^*}$, $u_{j^*} \; \Phi_n \; u_{k^*}$ we infer to

$$(u_{i^*} \oplus u_{j^*}) \; \Phi_n \; u_{k^*} \; .$$

Now it is easy to see that $i^* \oplus j^* = (i \oplus j)^*$, so $(u_{i^*} \; \Phi_n \; u_{j^*}) \; \Phi_n \; u_{k^*}$ implies /actually means/

$u_{i^* \oplus j^*} \; \Phi_n \; u_{k^*}$ implies

$u_{(i \oplus j)^*} \; \Phi_n \; u_{k^*}$ implies /by Lemma 1./

$u_{i \oplus j} \; \Phi_{n+1} \; u_k$ implies /means/

$u_i \oplus u_j \; \Phi_{n+1} \; u_k$

The remaining cases between

$$i, j, k < 2^n \quad \text{and } i, j, k \geq 2^n$$

can be handled in a similar fashion, especially taking into account the symmetry properties and Lemma 2 of the ring sum operation.

Having finished with the preparations we state the following THEOREM. For an arbitrary set $S \subset U_n$ the following conditions are mutually equivalent:

(i) S is $\wp_n$-closed

(ii) S is an intersection of U_n-generators

(iii) S is a boolean space .

Proof:

(i) implies (ii). See Fay /1973/ Theorem.
(ii) implies (iii). Every U-generator is BS by Lemma 5. The intersection of two BS's is a BS again by Lemma 4. So if S is a U-generator, then it is a boolean space.
(iii) implies (i). Let S be a boolean space. All we have to show is /for a fixed but arbitrary n/

$$\varphi_n(S) = \Phi_n\,(\Phi_n(S)) \subseteq S \tag{1}$$

for the opposite inclusion is well-known. /Szász, 1963.p.68/

Let correspondingly

$$x \in \varphi_n\,(S). \tag{2}$$

It is to be shown that $x \in S$.

(2) means, by (1), that for any $z \in \Phi_r\,(S)$.

$$x\ \Phi_n\ z \tag{3}$$

Let

$$S = \{s_1, s_2, \ldots, s_k\}\ ,\ k<2^n \tag{4}$$

and consider

$$y_i = x \oplus s_i \quad \text{for} \quad i=1,2,\ldots,k \tag{5}$$

Let

$$Y = \{y_1, y_2, \ldots, y_k\}\ .$$

First, we state, that

$$\varphi_n(S) \subset Y\ . \tag{6}$$

Indeed, S being a boolean space /according to Lemma 3/, one of its elements must be zero, therefore $x \in \varphi_n(S)$ implies /by (5) and by Lemma 2/:

$y_i \oplus s_i \in \varphi_n(S)$. But if $s_i=0$, than $x=y_i \in Y$. i.e. $x \in \varphi_n(S)$ implies $x \in Y$.

Secondly, we state that Y is a boolean space indeed for any

$$s_i \in S$$

$$s_i \; \Phi_n \; z \quad \text{with } z \in \Phi_n(S). \tag{7}$$

Now (3) and (7) implies, by Lemma 5 that

$$x \oplus s_i \; \Phi_n \; z \quad \text{for any} \quad z \in \Phi_n(S) \text{ and } \quad s_i \in S. \tag{8}$$

In other words, taking (5) for any $z \in \phi_n(S)$ and for any

$$x_i \in Y \text{ , we get}$$

$$y_i \; \Phi_n \; z \quad . \tag{9}$$

Applying Lemma 5 to (9) we get that for any y_i, $y_j \in Y$ and for any $z \in \Phi_n(S)$

$$y_i \oplus y_j \in \Phi_n(z) \; ,$$

i.e. /by (6)/ $y_i \oplus y_j \in \bigcap_{z \in \Phi_n(S)} \Phi_n(z) = \varphi_n(S) \; Y,$

$$y_i \oplus y_j \in Y \; .$$

This means that Y is a boolean space.

Now, by Lemma 3, we know that one of the y_i values must be zero:

$$0 = y_i = y \oplus s_i \quad \text{for any} \quad i \in \{1,2,\ldots,k\}.$$

This implies by Lemma 3 that

$$x = s_i \in S \; .$$

By this theorem it is quite easy to obtain all the φ_n-closed subsets of a given set U_n with a relation $\Phi_n(\subset U_n \times U_n)$.

All we have to do is just to find out all the sums yielding zero, methodically. The method is quite straightforward so it is enough to illustrate it by an example. Let us, by way of an example, produce all the φ_4-closed subsets of the set

$$U_4 = \{u_o, u_1, u_2, \ldots, u_{15}\} .$$

$u_o, u_1, \ldots u_7$ are trivially closed.

In addition to the trivially closed U_4 every closed subset must obviously contain $2^{4-2} = 4$ or $2^{4-1} = 8$ elements. These are generated by pairs or triples of elements from $\{u_o, u_1, \ldots, u_{15}\}$. Omitting u_o and working only with the indices the main part of the algorithm goes as follows.

Selecting pairs

First step:
Write equation

$$1 \oplus 2 = 3$$

infer that $S_1 = \{1, 2, 3\}$ is closed.

ℓ-st step:

If $S_{\ell-1} = \{i_{\ell-1}, j_{\ell-1}, k_{\ell-1}\}$ is closed

with $i_{\ell-1} \leq j_{\ell-1} \leq k_{\ell-1} = i_{\ell-1} \oplus j_{\ell-1}$

take the next /lexicographically/ pair (i_ℓ, j_ℓ) to $(i_{\ell-1}, g_{\ell-1})$ such that

$$i_\ell < j_\ell < k_\ell = i_\ell \oplus j_\ell \quad (i_{\ell-1} < j_{\ell-1}) .$$

Infer that

$$S = \{i_\ell, j_\ell, k_\ell\} \text{ is closed.}$$

Table I shows the actual steps for selecting pairs in case $n = 4$.

Selecting triples can be worked out in a quite similar fashion. Table II shows data for n=4. These are the same as in Abraham et al /1968/.

It is true, that our relation family Φ_n represents only a narrow family of finite geometries /namely those above GF (q) with $q=2^n$/ we do not know how to get a relation Φ_m^p for GF (q) with $q=p^m$, p prime, in general, such that finite projective geometry above GF (q) will consist of all the $\wp_m^p$-closed subsets of a set U_m^p where closure $\wp_m^p$ is induced by the relation Φ_m^p /$\subset U_m^p \times U_m^p$/.

REFERENCES

[1] Abraham, G.T., Ghosh, S.P. and Ray-Chaudhuri D.K.: File Organization Schemes Based on Finite Geometries, Information and Control 12, 1968. pp. 143-163.

[2] Bose, R.C.,Abraham, C.T., Ghosh, S.P.: File Organization of Records with Multiple-Valued Attributes for Multi-Attribute Queries. /Chapter 16 of Combinatorial Mathematics and Its Applications, Proceedings of the Conference, held at the University of North Carolina at Chapel Hill, April 10-14, 1967. /R.C.Bose and T.A.Dowling, eds/. The University of North Carolina Press Chapel Hill N.C.

[3] Fay,G.: An Algorithm for Finite Galois Connections, Technical Report, KGM ISZSZI, Hungary, 1973. VIII.15.

[4] Szász, G.: Introduction to Lattice Theory. The Publishing House of the Hungarian Academy of Sciences Budapest, and Academic Press New York and London, 1963.

[5] Herman, G.T.: Oral Communication, 1973.

TABLE I.

Determination of all the closed subsets, having 3 nonzero elements, of the set

$$\{0, 1, 2, \ldots, 15\}$$

Step	Equation	Closed set
1	$1\oplus 2 = 3$	1, 2, 3
2	$1\oplus 4 = 5$	1, 4, 5
3	$1\oplus 6 = 7$	1, 6, 7
4	$1\oplus 8 = 9$	1, 8, 9
5	$1\oplus 10 = 11$	1, 10, 11
6	$1\oplus 12 = 13$	1, 12, 13
7	$1\oplus 14 = 15$	1, 14, 15
8	$2\oplus 4 = 6$	2, 4, 6
9	$2\oplus 5 = 7$	2, 5, 7
10	$2\oplus 8 = 10$	2, 8, 10
11	$2\oplus 9 = 11$	2, 9, 11
12	$2\oplus 12 = 14$	2, 12, 14
13	$2\oplus 13 = 15$	2, 13, 15
14	$3\oplus 4 = 7$	3, 4, 7
15	$3\oplus 5 = 6$	3, 5, 6
16	$3\oplus 8 = 11$	3, 8, 11
17	$3\oplus 9 = 10$	3, 9, 10
18	$3\oplus 12 = 15$	3, 12, 15
19	$3\oplus 13 = 14$	3, 13, 14
20	$4\oplus 8 = 12$	4, 8, 12
21	$4\oplus 9 = 13$	4, 9, 13
22	$4\oplus 10 = 14$	4, 10, 14
23	$4\oplus 11 = 15$	4, 11, 15
24	$5\oplus 8 = 13$	5, 8, 13
25	$5\oplus 9 = 12$	5, 9, 12
26	$5\oplus 10 = 15$	5, 10, 15
27	$5\oplus 11 = 14$	5, 11, 14
28	$6\oplus 8 = 15$	6, 8, 15
29	$6\oplus 9 = 14$	6, 9, 14
30	$6\oplus 10 = 12$	6, 10, 12
31	$6\oplus 11 = 13$	6, 11, 13
32	$7\oplus 8 = 15$	7, 8, 15
33	$7\oplus 9 = 14$	7, 9, 14
34	$7\oplus 10 = 13$	7, 10, 13
35	$7\oplus 11 = 12$	7, 11, 12

TABLE II.

Closed subsets of $\{0, 1, \ldots, 15\}$ containing seven nonzero elements

Basis	Closed set
1, 2, 4	1, 2, 3, 4, 5, 6, 7
1, 2, 8	1, 2, 3, 8, 9, 10, 11
1, 2, 12	1, 2, 3, 12, 13, 14, 15
1, 4, 8	1, 4, 5, 8, 9, 12, 13
1, 4, 10	1, 4, 5, 10, 11, 14, 15
1, 6, 8	1, 6, 7, 8, 9, 14, 15
1, 6, 10	1, 6, 7, 10, 11, 12, 13
2, 4, 8	2, 4, 6, 8, 10, 12, 14
2, 4, 9	2, 4, 6, 9, 11, 13, 15
2, 5, 8	2, 5, 7, 8, 10, 13, 15
2, 5, 9	2, 5, 7, 9, 11, 12, 14
3, 4, 8	3, 4, 7, 8, 11, 12, 15
3, 4, 9	3, 4, 7, 9, 10, 13, 14
3, 5, 8	3, 5, 6, 8, 11, 13, 14
3, 5, 9	3, 5, 6, 9, 10, 12, 15

Subscription price: Hfl 82,— /Volume