[*International Conference on Machine Translation of Languages and Applied Language Analysis,* National Physical Laboratory, Teddington, UK, 5-8 September 1961]

A TECHNIQUE FOR CONSISTENT SPLITTING OF RUSSIAN WORDS

by

D.W. DAVIES and A.M. DAY

(National Physical Laboratory of England)

A mechanical dictionary of Russian words contains a number of ***entries.*** Each entry is identified by its Russian word, and contains certain output data concerning that word. The simplest form of dictionary would have a separate entry for every variant of a word, for example, there would be entries for each inflected form. Economy of space usually dictates that the dictionary entries be related not to the words themselves, but to smaller parts out of which the words are built.

If a word is regarded as built from two parts, the storage of these separately in the dictionary leads to economy only if

1. These parts are both used to build several words.
2. The output data required for the words can be formed from a reasonable quantity of data stored for their parts.

The inflection of Russian Nouns, Adjectives and Verbs (Including participles) is a type of word-building which can give considerable dictionary economy. The parts of the words we use in this case are:

A ***stem,*** which ideally is invariable but may have some changes in practice.

An ***ending,*** which for each stem, is taken from a small set of endings. The list of all the endings used in Russian is not very large.

There are in Russian many other types of word-building which are regular in their operation, but the case for using them in practice to obtain dictionary economy is not so clear as it is for the inflections mentioned above. In other languages, different types of word-building may prove to have the necessary regularity. For example, in English the forma-

tion of adverbs from adjectives is regular and is applied to suitable newly coined adjectives, therefore it could lead to economy in an English dictionary.

We shall assume, then, a dictionary in which each *stem* of a noun, adjective or verb has a separate entry, and the entry has suitable coded information to indicate the ***endings*** it can associate with. The nature and use of this coded information about endings is the subject of two other papers.[1]

When a word is looked up in this dictionary, it is identified with a certain stem and ending, therefore at some stage in the process the word is ***split***. The process would be very simple if it were possible to achieve this split at the very first stage, simply by examination of the given word. We have available a list of all possible endings, but the largest ending that can be identified is not necessarily the correct one. For example, the split shown by the vertical bar in the word AT|OM is plainly incorrect, although -OM is a valid ending. The correct split in this example should be shown as

ATOM|*

where * is used to indicate the null ending. (The majority of splitting difficulties are caused by the null ending.)

If we wish to look up the words from a block of text in one pass of a dictionary stored on magnetic tape, these words must be arranged in dictionary-sequence, which we shall suppose to be alphabetic sequence. But a dictionary in which the entries refer to ***stems*** must be in alphabetic sequence of stems, whereas unsplit text words can only be in alphabetic sequence of whole words. The difference between these sequences is not great, in general, and there are schemes which achieve look up without splitting beforehand. The search through the dictionary is then not unidirectional. But we propose to consider in this paper a technique in which the text words are placed in exactly the same sequence as the dictionary entries, so that a unidirectional search through the tape is sufficient.

---

1 (Davies 1960, McDaniel and Whelan, this conference.)

The Idea of Consistent Splitting

It is the variable ending which causes the difficulty of getting text words into dictionary sequence. If this ending is removed there is no further difficulty. We have said that universally correct splitting is impossible, but there is an alternative, which is to make a split which is always at the correct point or earlier in the word. The variable ending is certainly removed, and perhaps a small part of the stem as well.

In this scheme, the split made before consulting the dictionary is not necessarily the correct split into stem and ending. For example, in the splitting scheme which we actually use, the word МЕТАЛЛ is split, in its various inflected forms

МЕТАЛ|Л
МЕТАЛ|ЛА
МЕТАЛ|ЛУ

etc.

This example illustrates the principles by which the splitting rules were devised. The forms МЕТАЛЛ and МЕТАЛЛА in any simple splitting scheme would be wrongly split because the endings —Л, -ЛА, occur in the past tense of verbs. It is not possible to know before looking that МЕТАЛЛ is not a verb, and, although more elaborate rules would certainly recognize the error, we choose the simplest rule, according to which —Л and -ЛА are split off. Then the inflected forms of МЕТАЛЛ would give rise to two stems because МЕТАЛЛУ and the other forms would split 'correctly' after the true stem МЕТАЛЛ. The new idea is to arrange the splitting rules so that all the inflected forms of МЕТАЛЛ split in the same place, even though this is to the left of the correct split. We refer to this as consistent splitting.

The dictionary entry for МЕТАЛЛ must show both the part МЕТАЛ which determines its place in the alphabetic sequence, and the full stem МЕТАЛЛ. These together may be regarded as a ***split entry*** МЕТАЛ Л
The process of consulting the dictionary is as follows.

1. A splitting routine is applied to each text word in turn, resulting in a pseudo-stem and pseudo-ending, example:
МЕТАЛ|ЛУ

2. The words are sorted into alphabetic sequence of pseudo-stems.

3. These sorted items are compared with the dictionary, the common sequence being alphabetic sequence of pseudo-stems, then:

3(a) Whenever a pseudo-stem matches the pseudo-stem *of* an entry, a check is then made to see if the whole of the entry is contained in the given text-word. If it is, what remains of the text word is the ending, and this is checked for compatibility with the inflection data. If it is compatible, the text word has been identified in the dictionary, and we do not continue our description of the process beyond this point.

3(b) If several identical pseudo-stems are encountered in the dictionary these must all be checked against text words with this pseudo-stem.

We have described the aim of our splitting procedure as 'consistent splitting' which means that all the inflected forms of a given word split down to the same pseudo-stem. This requirement is satisfied by splitting procedures in which large parts of the true stem are chopped off in the pseudo-ending. In the limit, it is consistent to split all words in front of their first letter. The disadvantage of excessively long pseudo-endings is that they create artificial pseudo-stem homographs. A large set of homographic pseudo-stems makes the process 3(b) excessively long, and effectively throws some of the sorting phase of the process into part (3). For these reasons, the splitting routine should be made to approximate as far as possible to correct splitting.

The Use of Cross-references for further Economy

A single Russian noun, adjective or verb may give rise to more than one dictionary stem either because it has a variation in its true stem or because the splitting is not completely consistent, and gives rise to more than one pseudo-stem. The entries for a given word contain almost the same output data. It is therefore wasteful to store several full length entries, and all but one of these could be replaced by cross references.

When the multiple entries are due to an unavoidable inconsistency in splitting, it usually happens that most of the inflected forms split with the longest pseudo-stem. This, fortunately, occurs last in the dictionary. The shorter pseudo-stems can be represented in the dictionary by small items which merely state how many letters to transfer from ending to stem in order

to obtain a pseudo-stem referring to the full entry.

An example of inconsistency in the splitting scheme we use is provided by the verb ПЛАТИТЬ. There are two pseudo-stems, ПЛА obtained from

ПЛА|ТИМ

ПЛА|ТИ

ПЛА|ТЯ etc.

and ПЛАТ from

ПЛАТ|ЯЩ|ИЙ

ПЛАТ|ИМ|ЫЙ

ПЛАТ|ИВШ|ИЙ etc.

The entry for ПЛА would merely be a cross reference to ПЛАТ. This could be denoted ПЛА|Т| where the two bars indicate the old and the new splits. The new split in this case happens to be a correct split, and this is often the case. A different example is the noun НАЛЕТ which gives rise to two pseudo-stems НАЛ and НАЛЕ neither of which is the correct stem. The cross-reference entry under НАЛ would read НАЛ|Е|Т. When the text word НАЛЕТЕ occurs it is split НАЛ|ЕТЕ and referred to the entry НАЛ. At this entry it is first verified that the whole entry НАЛ|Е|Т is contained in the given word, then the new split НАЛЕ|ТЕ is generated and the word is referred to the full entry under НАЛЕ|Т. Since it matches, the remainder -Е is the ending, and this is now checked against the inflection data.

Because cross-references of this kind always refer forwards, the newly generated splits can be held in a small store until the appropriate part of the dictionary arrives. The entries are usually close together, so the contents of the store would be small. This is important, because these items have to be kept in sequence, for ease of reference.

Some kinds of stem change can also be handled by this method of reference forwards, namely mobile vowels. This is described elsewhere (Davies 1960).

It is not necessary to keep full entries and cross-references in the

same file. An alternative technique is to keep cross-references in a separate file and run through this before going to the main dictionary. The outputs of the cross-reference file are sorted and merged back into the alphabetized text. The text words which gave rise to cross-references would be retained, in case there were full entries homographic to cross-reference entries.

A Scheme of Consistent Splitting

The development of a consistent splitting process is the main subject of an earlier paper (Davies 1960) which should be referred to for details. Only the principles are mentioned here.

There are three stages of splitting. The first stage is to remove the endings —СЯ or -СЬ, which are called 'zero-order endings'.

Whether these have been removed or not, the next stage is to remove any of a list of 'first-order affixes' from what remains. These are listed in ***Table 1***. The endings which must be identified are listed, and, for each one, the split which must be made is shown by the bar. If no bar is shown, the whole of the identified ending is split.

Endings which must be identified but are not completely split off, occur in two ways. Firstly, there are endings like —Ч|АТ where the reason that —АТ is accepted as an ending rather than —ЯТ is the presence of Ч, with its spelling peculiarities. We require the presence of Ч but do not split it. Secondly there are endings like -АНИ|Е which are productive word-building suffixes. Without this special rule, the endings would be split -АН|ИЕ which is inconsistent with the remainder of the declension. In these cases, the split — |ИЕ is first made, but our program always looks further for the presence of -АН—, and if it finds it, it changes the previously assumed split to —|Е.

The next stage of splitting is the removal of the largest of a list of 'second-order affixes' from the end of what now remains. This process is carried out whether or not any zero- or first-order affix has been removed. The second-order affixes are listed in ***Table 2***.

Broadly speaking, the second-order affixes are those which are added to verbal stems to make participles, being followed in general by adjectival first-order affixes. Various restrictions can immediately be seen in the application of affixes from these lists. But to apply these restrictive rules, it turns out, generally creates inconsistency. There is

one exception in that first-order affixes which can ***only*** occur at the end of the inflected forms of verbs cannot be preceded in these forms by second-order affixes. We adopted this restriction because it improved the consistency of splitting of certain nouns. It is shown in ***Table 1,*** by the use of a double bar as in ||ИТЪ and ||ШИ, which indicates that if this split is made, no second-order split should then occur. An example of consistency due to this rule is

ДОЛОМ || ИТ

ДОЛОМ | ИТ| У

(The first- and second-order splits are both shown by bars in this example.)

When a participle (other than masculine singular short form) is subjected to this split procedure, it will always have its adjectival and particlpal affixes correctly identified as first- and second-order splits. Other words than participles, while being consistently split, may have their endings removed as either first- or second-order splits or both.

Several features of ***Tables 1 and 2*** may seem anomalous, for example the entries ||НЕН in ***Table 1,*** and ИЛ in ***Table 2.*** The reasons for these oddities are that they improve consistency in certain common cases. The earlier paper (Davies 1960) describes the derivation of these tables in detail.

TABLE 1

First Order Affixes

А	И	||Л	У
||ЛА	||И	||ИЛ	ЕМУ
||ИЛА	||ИЛИ	АМ	ОМУ
В	АМИ	ЕМ	АХ
ЕВ	ИМИ	ИМ	ИХ
ИВ	ЫМИ	ОМ	ЫХ
ОВ	ЯМИ	ЫМ	ЯХ
СТВ|	||ТИ	ЯМ	Ы
Е	||ШИ	||НЕН	Ь
ЕЕ	||ВШИ	||ЕНЕН	||ТЬ
ИЕ	||ИВШИ	О	||ИТЬ
ВИ|Е	И	ЕГО	||ЕШЬ
АНИ|Е	ЕЙ	ОГО	||ИШЬ
ЕНИ|Е	ИЙ	||ЛО	Ю
ЯНИ|Е	ВИ|Й	||ИЛО	УЮ
ТИ|Е	АНИ|Й	||Т	ЬЮ
ОЕ	ЕНИ|Й	Ж||АТ	ЮЮ
||ЕТЕ	ЯНИ|Й	Ч||АТ	Я
||ИТЕ	ТИ|Й	Ш||АТ	АЯ
||ЙТЕ	ЦИ|Й	Щ||АТ	ЯЯ
||ЬТЕ	ОЙ	||ЕТ	
ЫЕ	ЫЙ	||ИТ	
		||УТ	
		||ЮТ	
		||ЯТ	

TABLE 2

Second Order Affixes

Л	Н	Т	АЩ
ИЛ	ЕН	ИТ	УЩ
ЕМ	НН	Ш	ЮЩ
ИМ	ЕНН	ВШ	ЯЩ
ОМ		ИВШ	

The Tree Structure of the Split

The process of splitting a word consists basically of the selection from a table of the largest affix matching the ending of the word. In practice we carry out this procedure twice, using ***Tables 1 and 2*** *in* succession.

Certain features of the tables which we exploit will be described, initially for ***Table 1*** only, although the same techniques apply to the use of both tables.

Many of the final letters in ***Table 1*** are seen to be shared, in some cases several times. In fact while there are 89 endings in the table, the number of different final letters is 16 – А В Е И Й Л М Н О Т У Х Ы Ь Ю Я. This sharing applies also to several final pairs and final triads of letters. As an example, the letter Е is an affix itself and is also a final letter of several other affixes, one of which is ИЕ; ИЕ is a final pair of letters common to a number of affixes, and so on.

This feature of sharing can best be illustrated in the form of a tree, a portion of which is shown in ***figure 1***.

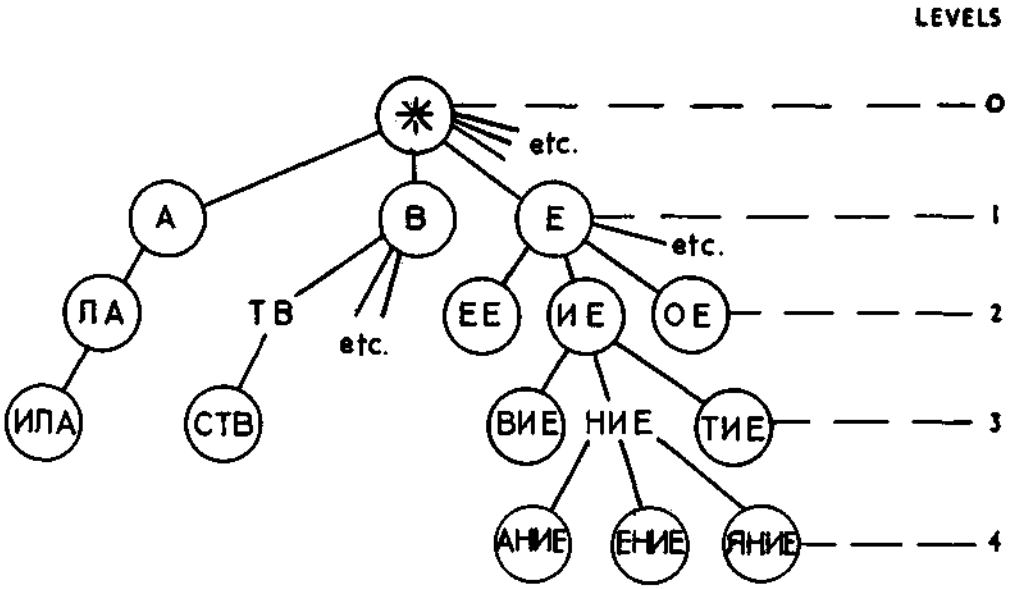

Fig. 1

This tree structure will be used to describe the splitting process. Most of the elements in the tree are shown enclosed in a ring. These are the ones which correspond to valid endings - i.e. endings in ***Table l.*** The unringed elements, such as НИЕ are merely intermediate steps to valid endings.. Note that the null ending is a valid one, hence it is ringed. If from an element α in the tree lines lead downward to a set of elements on the next level, this set of elements is here called the family of a and if β *is* a member of the family of a we say that α is the unique parent of β, e.g. the family of ИЕ consists of ВИЕ, НИЕ and ТИЕ.

Three Possible Ways of Programming the Split

We describe three ways in which the splitting technique, summarized in ***Tables 1 and 2,*** can be carried out, but it is the third of these, (c), that we actually use. The three alternatives are all of interest, and each has advantages which could be decisive in certain applications.

(a) ***Searching One List***

Perhaps the simplest method of affix splitting is to arrange the table

in the form of a list whose items are the 89 endings. Against each ending is inserted the number of letters to split, pertinent to that ending. This number we refer to as the split number. A portion of the list might be:-

ЯНИЕ	1
ТИЕ	1
ОЕ	2
ЕТЕ	3

The list is then searched and the largest ending which will match is found and its corresponding split number retained. An obvious expedient is to arrange the list in order of size of ending and allow the search to proceed from the largest endings to the smallest. The first ending to match is taken and the search is complete.

(b) ***A Fully-programmed Tree-Search***

This system utilizes a program exploiting and even resembling in its procedure the tree-structure of the affix table. Each affix in the tree is represented in the program by a set of instructions which compare the affix with the ending of the word to be split.

If W is the word to be split the program compares the final letter of W with the letters on level 1 of the tree. If no match is found the search ceases. In the event of a match, however, each member of the matching-letter's family on tree-level 2 is compared with the final pair of letters of W. A further match leads to comparison with the appropriate family on level 3, etc. The process continues until, at a certain level, either

(1) no match is found with the members of a family,
or
(2) the matching ending has no family.

In either case the outcome of the search is taken to be the last ending to match and its corresponding split number. This method is faster than method (a) but occupies considerably greater program space.

(c) ***A Tree-Search Guided by a Structured List***

A structured list, which reflects the structure of the tree, is used in

this system. The list guides a programmed search from level to level causing it to jump unrewarding sections of the list. Against each ending, in addition to the split number, is the address of the head of the list forming its family which occupy successive locations in the store. Thus if a match is made, the address against the matching ending guides the search straight to the family of the ending.

The remainder of the paper is concerned with this particular method of splitting. Its speed of operation is not so great as method (b) because more time is spent on organizational detail than on actual search. However, the instruction and data storage space it occupies is less than that of method (b). Compared with method (a) it may occupy a little more space but is considerably faster.

Description of a Simple Tree by a Structured List

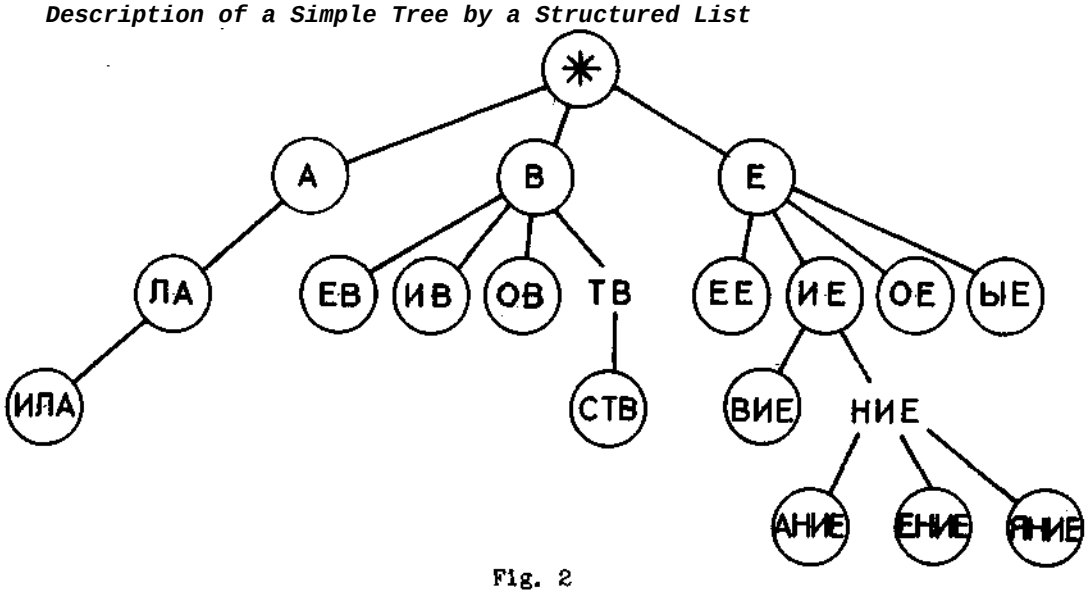

Fig. 2

The notation for describing the tree by a structured list is illustrated by the small tree of ***figure 2*** (which is part of the actual tree used), and the corresponding list of ***figure 3,*** which would occupy 19 store locations.

ADDRESS	(a)	(b)	(c)	(d)	(e)
1	А	1	4		А
2	В	1	5		В
3	Е	1	9	1	Е
4	Л	2	13	1	ЛА
5	Е	2	0		ЕВ
6	И	2	0		ИВ
7	О	2	0		ОВ
8	Т	1	14	1	ТВ
9	Е	2	0		ЕЕ
10	И	2	15		ИЕ

ADDRESS	(a)	(b)	(c)	(d)	(e)
11	О	2	0		ОЕ
12	Ы	2	0	1	ЫЕ
13	И	3	0	1	ИЛА
14	С	0	0	1	СТВ
15	В	1	0		ВИЕ
16	Н	2	17	1	НИЕ
17	А	1	0		АНИЕ
18	Е	1	0		ЕНИЕ
19	Я	1	0	1	ЯНИЕ

Fig. 3

(Column (e) is not contained in the list used but
is merely shown in the diagram to facilitate description)

Each item of the list contains four parts:-

(a) The letter which is currently being tested (Test letter). This letter is the first letter of the full ending shown in column (e) of the list, but the full ending is not included in the list stored in the computer, because all the letters except the current one have already been identified at earlier levels of search.

(b) The number of letters to be split off as ending (Split Number). If the test letter agrees with the appropriate letter of the word this number gives the amount of split - assuming that the further search which follows does not produce any larger matching ending.

(c) The next address reference. This is the address of the first member of its family. A zero in this column however indicates that the ending corresponding to this entry has no family and is a terminal point of the tree. Thus this ending is the largest valid ending and the search is terminated.

(d) End of list marker. A 1 in this column indicates that the ending corresponding to this entry is the last one of a short list constituting a family. If the search has proceeded up to and including such an entry then the parent of the family is taken as the largest matching ending.

This list, comprising columns (a), (b), (c) and (d) completely describes the tree, and forms the data necessary for an efficient search.

Elements of the tree such as ТВ and НИЕ which are not themselves endings are distinguished in the list only by the values of their split number. Thus the split number of entry 16 of the list which corresponds to НИЕ is 2, showing that only the ИЕ previously identified is a possible ending. The split number of АНИЕ in entry 17 is 1, showing that if АНИЕ is identified, only the Е is to be split.

Before the programme which uses this list is described, we must explain the need for further data to be included in the list.

Program for a Simple Tree including 'Affix Identifier' as an Output

A Russian word consists of a stem giving the root-meaning of the word plus an affix, whose role is to indicate variously, case, gender, number, etc. The role of an affix may be codified but since certain affixes have numerous roles the codified information may be extensive.

However, if all the coded information is kept in a store the address in the store of the information pertaining to a particular affix will serve to identify the affix temporarily until the full information concerning it is required. Hence we will refer to this address as the 'affix-identifier'.

The output from a splitting program should be therefore not only the split number but also the 'affix-identifier' of the split affix. Thus for the list in ***figure 3*** to be really useful to a program it should contain against each affix a further column for output, namely an 'affix-identifier' column. A splitting program, utilizing such an extended list to split from a word the largest possible affix and to obtain the appropriate affix-Identifier, would be of the form in ***figure 4.***

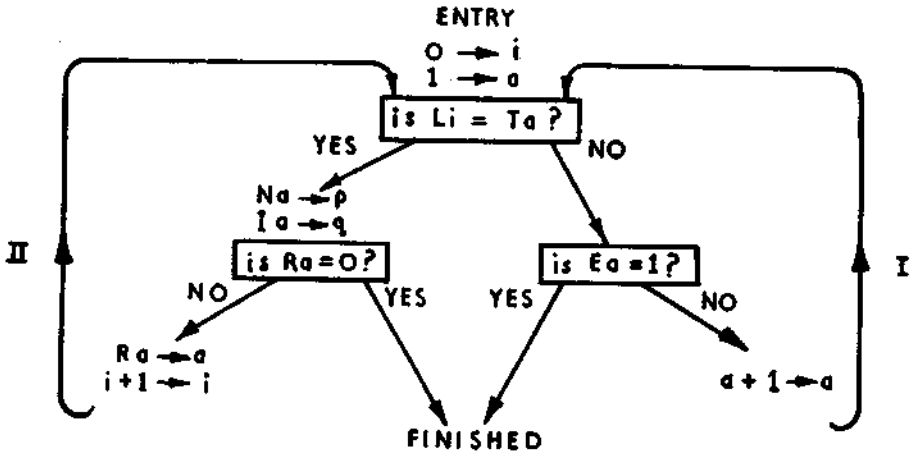

Fig. 4

If W is the word being split

L_i = current letter of W being examined

L_0 = last letter of W

L_1 = next to last, etc.

a = address in list

Data in list:-

Ta = test letter

Ra = next address reference

(Ra = 0 means this item has no family)

Na = number of letters split

Ia = affix-identifier

Ea = end of a family list

Finally:- p contains split No.

q contains affix-identifier.

An excursion round the right-hand loop I of the diagram is equivalent to a horizontal movement across the tree from member to member of the same family, while the left-hand loop excursion II constitutes a vertical movement down the tree from parent to family.

The Development for Zero-, First- and Second-order Splits

The description of program techniques for splitting has so far dealt merely with the identification and splitting, etc. of the largest possible affix from one tree or list of affixes. We are in fact using three stages of splitting, each with its own list of affixes, and which we term zero-, first-, and second-order splitting stages respectively.

The zero-order stage is trivial because its list consists merely of the two reflexive endings -СЬ and -СЯ. If one of these is identified as the ending of the word a note is made that the word is reflexive and the affix is split off the word. What remains is passed onto the first-order stage of splitting.

The first-order stage is almost identical to the program described above, but with a considerably enlarged list structure. The largest possible ending from the first-order list is now split from the word, the affix-identifier is stored and the remainder of the word is finally passed to the second-order stage.

Identical to the first-order stage except of course for its list, the second-order stage, yielding the appropriate split and a further affix-identifier, completes the process.

Though their lists are different, the actual procedures for the first- and second-order splittings are identical and so the one program may serve for both. In fact when the first-order splitting is finished the program is re-entered but now guided by the second-order list.

However, in our rules for consistent splitting we have decided that where certain first-order affixes are Identified and split from the word, an attempt at second-order splitting must not occur. This additional guiding information is inserted in the list against each ending.

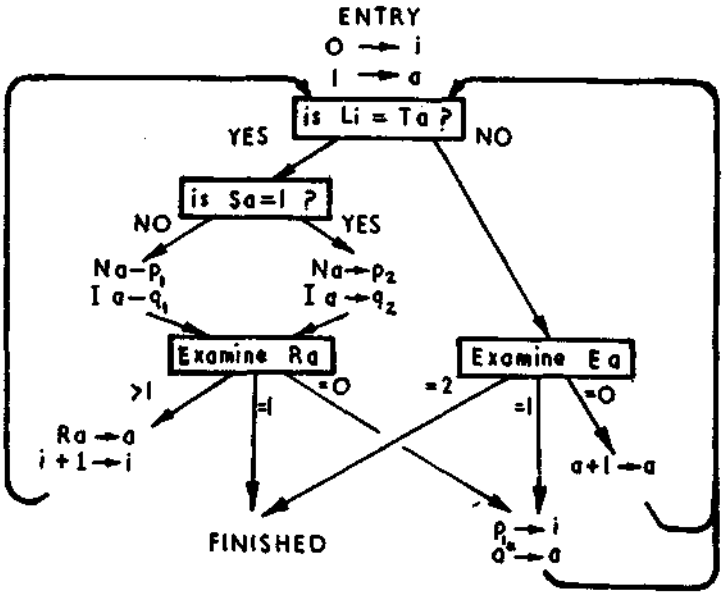

Fig. 5

L_i = current letter being examined of word under test

L_0 = last letter

L_1 = next to last

etc.

a = address in list

a* = address of first item in second-order list

Data in table:-

Ta = test letter

Ra = next address reference

R = 0 means end of 1st-order, go to 2nd-order split

R = 1 means end of process, 2nd-order not allowed

Na = number of letters split

Ia = affix-identifier

Sa { S = .0 for first-order split
S = 1 for second-order split

Ea { E = 0 end of family list not yet reached
   { E = 1 end of family list, go to second-order split
   { E = 2 end of family list, second-order split not allowed

Finally:- P_1 contains split No., q_1 contains affix-identifier for first order; P_2 contains split No., q_2 contains affix-identifier for second-order.

figure 5 *i*llustrates a program to perform first- and, where allowed, second-order splitting. It is seen to be merely an extension of that shown in ***figure 4***. On completion of the first-order split the program is made to recommence searching at the beginning of the second-order list. This is done by the two instructions $P_1 \rightarrow i$ and $a^* \rightarrow a$ at the bottom of the flow diagram. The first of these restores the index 1 which determines the current letter of the word being split. The letter following the first-order split is the next one to be examined. The second instruction directs attention to the address a* of the second-order table. The total output in most cases consists of the two split numbers and the two affix identifiers plus a single digit which indicates whether a zero-order affix was found.

Information needed for Correction of Splitting at a Later Stage

For the reasons described earlier, the splitting procedure cannot be always correct in its results. The correct split is, however, obtained, either by a cross-reference in the dictionary or by a suitable notation in the dictionary entry.

It is thus necessary to obtain the affix-identifier for the correct affix. As we described the procedure of splitting, only one affix-identifier for each order of splitting was retained. In practice we retain the affix-identifier in each entry of the list which is used in building up the full affix.

Consider for example the splitting of the word HAJIETE. This is first matched with the ending E which is a valid ending, then with the entry TE, which is not valid, then with the entry ETE, which is a valid ending and does not admit a second-order split.

The affix-identifiers of E and ETE are 15 and 91 respectively. A word is constructed by the splitting program which has space for six

affix-identifiers, one for each possible letter of a maximum size of split. The present example fills these six cells as follows:

No. of letter split:	1	2	3	4	5	6
affix-identifier:	15		91			
affix	E		ETE			

The indicated split number is 3, so the appropriate affix-identifier would be 91. During dictionary look-up, it is found that the correct split is HAJET|E and at this stage the affix-identifier 15 is known to be correct.

REFERENCES

DAVIES, D. W. "The Organization of a Russian-English Stem Dictionary on Magnetic Tape", *Language and Speech*, 1960, **3**, part 4, pp. 193-222.

McDANIEL, J. and WHELAN, S. "The Grammatical Interpretation of Russian Inflected Forms using a Stem Dictionary". This Conference).