[*Mechanical Translation*, vol.4, no.3, December 1957; pp. 54-58]

A Type of Program for Mechanical Translation

J. P. Cleave, University of Southampton, Southampton, England*

A program for the mechanical translation of a limited French vocabulary into English was constructed for operation on the computer APEXC. Its principal features were an improved routine for dictionary look-up, and an organization permitting systematic incorporation of additional subroutines. A program for syntactic processing was constructed but was too large for the available storage space. It examined preceding and following items — stems or endings — in order to choose correct equivalents, and used a dictionary of syntactic sequences or structures to effect local word-order change.

APEXC

The computer has a magnetic drum store with 1024 locations arranged in 32 tracks each of 32 locations. Each location contains 32 bits. Any location can therefore be specified by an address of 10 bits. Both data and instructions are stored on the drum.

An instruction consists of 32 binary digits and specifies an operation (function), the 10 bit address of an operand contained in the store and the address (10 bits) of the next instruction, which again is contained in one location in the store. The arrangement of the digits of an instruction is shown below (Fig. 1).

APEXC has one branch (jump) instruction discriminating between positive (or zero) and negative.

The following abbreviations will be used:

O_x	operand address (X-address) of an instruction O.
O_y	next instruction address (Y-address) of O.
$(O_x)_{ls}$	least significant digit of O_x (i.e., digit 10).
$(O_y)_{ms}$	most significant digit of O_y (i.e., digit 11).
(z)	contents of the location whose address is z.

Dictionary Subroutines

The dictionary procedure is best explained by considering a simplified example with a dictionary of 16 positive entries stored in increasing numerical order in locations 1, 2, 3, ... 16. Suppose W is a word, known to be in the dictionary, whose address in the dictionary is required.

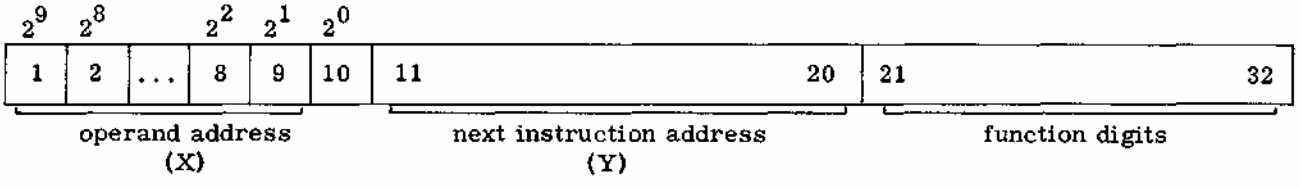


Figure 1

---

* This paper is a report of work done in cooperation with Dr. A. D. Booth and Mr. L. Brandwood at the Computational Laboratory, Birkbeck College, London.

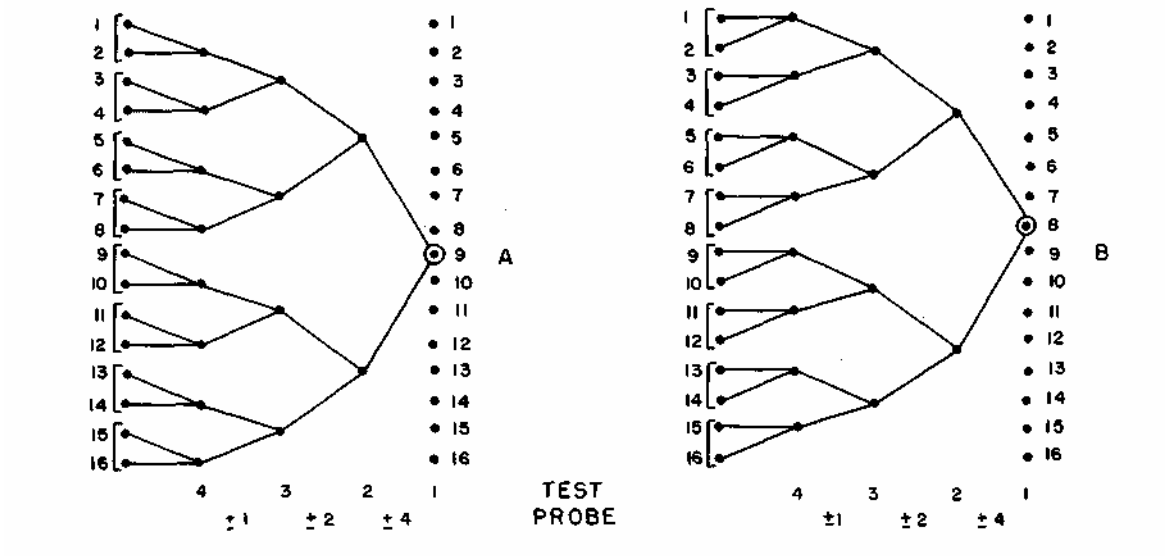


Figure 2

The bracketing procedure[1] requires us to start in the middle of the dictionary, either at 8 or 9. Suppose 8 is chosen; the procedure for 9 is analogous (see Fig. 2).

An "operation" consists of forming W-(y) by means of a subtraction instruction O. If the result is positive, a "probe-number" p is added to O_x, if negative it is subtracted, p is then divided by 2.

The first operation is on (8) (i.e., $O_x = 8$) with $p = 2^2$. After the operation $O_x = 12$ or 4 (i.e., $O_x = 8 + 2^2$ or $8 - 2^2$), the new probe-number is $p = 2^1$.

The second operation gives a new probe-number of 2^0. The third test, therefore, shows W to be in one of the 8 sets of 2 shown in the diagram.

The fourth operation is slightly different from those preceding. It can be seen that operations 1, 2, 3 each discriminate between two new addresses: the fourth discriminates between one new address and one that has been tested before. If we now examine the dictionary entry specified by O_x at the beginning of operation 4, it can be seen that W is either in O_x or $O_x + 1$. (If the initial location had been 9, the alternatives would be O_x and $O_x - 1$.) Hitherto, dictionary subroutines we have used counted the number of operations performed and at the final operation tested O_x and its neighbor for identity with W. This latter test had to be synthesized and so required several instructions. This disadvantage can be eliminated if the final operation is similar to its predecessors.

Suppose operation 4 is similar to 1, 2, 3. At the conclusion of the third test $p = 2^{-1} = 1/2$. This is a '1' in $(O_y)_{ms}$. The X-addresses formed are shown in Fig. 3.

If the initial location is 9 and $(O_v)_{ms}$ prior to operation 3 is '0', the correct address of W in the dictionary will be formed in O_x. But O_v.. is the address of the next instruction to O in the dictionary routine and is altered by the addition of 2^{-1} to O_x to $O_v' = O_v + 2^9$, thus enabling a jump to occur at precisely the right moment in the sequence of operations. O_v' is the address of the first instruction of the routine following dictionary look-up. If the initial

1. Booth, A. D., "Use of a Computing Machine as a Mechanical Dictionary", Nature, vol. 176, Sept. 17th, 1955, p.565.

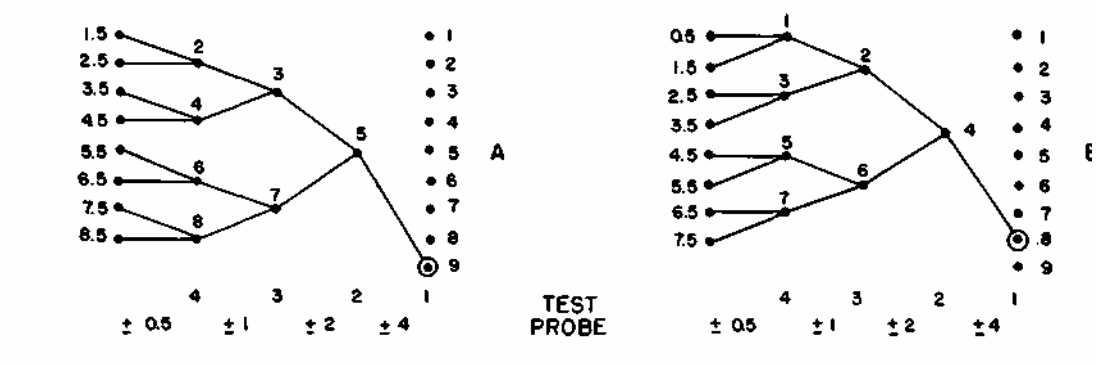


Figure 3

location is 8, W is located correctly only if $(O_y)ms = 1$ Here $O_y' = O_y - 29$

The efficacy of this method clearly depends upon the fact that $(O_x)_{ls}$ is next to $(Oy)_{ms}$ (see Fig. 1). This convenient arrangement now enables us to dispense with special arrangements for the final operation, counting the number of operations performed and special orders for jumping to the next sequence. The dictionary program now occupies only 11 locations: it was used in the MT program explained below.

If the W is not in the dictionary, then this method of dictionary look-up will select the greatest entry less than W.

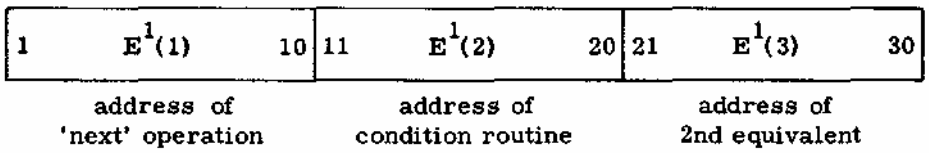
It might be supposed that a further increase of speed could be obtained if during each of the above operations a test for zero is made (i.e., identity between W and the dictionary entry). Suppose a dictionary of 2^n entries. One dictionary entry can be located during the 1st test, 2 during the 2nd, 4 during the 3rd, ... 2^{r-1} during the r^{th} , . . .; $2^{n-1}+1$ requires n tests. (The extra 1 is an entry that cannot be located by a zero test: in the examples of Fig. 2, either 1, or 16.) Assuming that each entry is equally likely to occur in a text, the average number of operations to locate a single word is

$$m = [1.1 + 2.2 + 4.3 + \ldots + r2^{r-1} + \ldots + (n2^{n-1} + n)] /2^n$$
$$= n - 1 + (1 + n)/2^n.$$

Thus if n is large only one operation is saved; the extra programming required in a test for zero is therefore not worth-while with a computer without this facility.

The Basic MT Program

All data to be "recognized" were, with a few exceptions, included in the main dictionary. The input routine compared sequences of symbols between "space" marks with the dictionary entries. This routine therefore had only to recognize a "space" symbol on the input tape. All punctuation marks, and the symbol for the end of text, were included as dictionary entries. Each dictionary entry D of the main- and ending-dictionaries was confined to one storage location and had two equivalents. The second of these, E^2, was the target language equivalent of the dictionary entry. In general E^2 occupied several locations. All "syntactical" operations were performed on the "first equivalents, " E^1 , each of which occupied only one storage location. Each E^1 was constructed uniformly and consisted of three sets of ten digits specifying addresses $E^1(1)$, $E^1(2)$, $E^1(3)$. (See Fig. 4.)



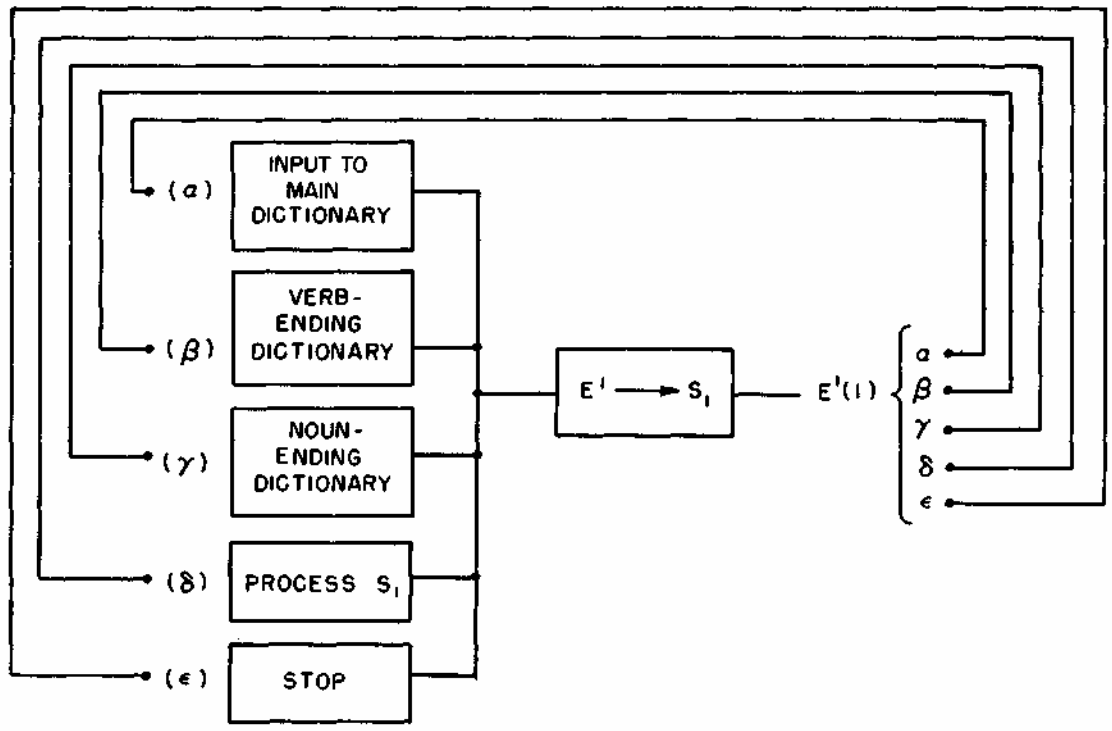
Structure of First Equivalent

Figure 4

The address $E^1(1)$ of E^1 was used in the following manner. After an input datum had been identified (either a complete word, or stem) its first equivalent E^1 was placed in a set S_1 of consecutive storage locations. $E^1(1)$ then specified the address of the first instruction of the next sequence of operations. Thus if a verb were compared with the main dictionary, E_i^1 of the stem was extracted and placed in location n of S_1. $E_i^1(1)$ then determined the next operation. $E_i^1(1)$ for a verb stem was β, the address of the first order of the routine directing the dictionary look-up procedure to the verb-ending dictionary. E_j^1 of the verb ending was then stored in location n + 1 of S_1. $E_j^1(1)$ was the address α of the input routine. Thus after the first equivalent of the verb stem had been stored in S_1, a new word was fed in to be compared with the entries of the main dictionary. The first equivalent of a full stop had address $E^1(1) = S$, the address of the initial instruction of a routine for processing the accumulated data in S. (Fig. 5.) $E^1(1)$ for an end-of-text symbol was ε, a stop order.

A program for processing the first equivalents was constructed but was found to be too large for the available storage space and was abandoned. The plan of this routine, however, will be stated.

The processing of S_1 consisted of carrying out in turn the operations whose first instructions were determined by the second address $E^1(2)$ of each first equivalent in S_1. These operations — condition routines — had two functions. The first was to examine, where necessary, equivalents preceding and following to determine whether $E^1(3)$ specified the correct second equivalent. The second function was to place a code number C corresponding to E in another series of locations S_2. Convenient sub-sequences of the code numbers in S_2 were then compared to a "structure-dictionary." Recognition of these sub-sequences resulted in a rearrangement of the order of the recognized



The Function of First Equivalents

Figure 5

C-sequence and the corresponding E^1 -sequence. The code-numbers were therefore assigned in such a manner that the sequences requiring rearrangement could be recognized distinctly. Although in most cases this assignment coincided with the usual classification of verb, pronoun, etc., there were some C which did not correspond to these categories. Thus donn was entered in the main dictionary, with 'give' as the target language equivalent. The condition routine for this entry assigned a code number ($verb_1$) to it. erons was an entry in the verb-ending dictionary. The condition routine determined by its first equivalent gave it a code number ($verb_2$). The second equivalent of erons was 'will'. Thus when donnerons occurred in the input text, the first equivalents of donn and erons were placed in consecutive locations in S_1. When the condition routines were operated, the code numbers ($verb_1$) and ($verb_2$) were placed in order in S_2. Following these routines the structure dictionary recognized the sequence ($verb_1$) ($verb_2$) as one requiring transposition. The corresponding data in S_1 were then transposed. Thus the final printing operation printed the target language equivalents of donn/erons in reverse order to yield 'will give'. This procedure was used to perform the pronoun-verb inversion.

The final stage of the program was a routine for printing the second equivalents. In the program which was put on APEXC the processing of S_1 was omitted so that the dictionary routines were immediately followed by the print routine. The print routine printed the contents of the addresses specified by the 3rd address of the first equivalents in S_1. Each location containing a second equivalent also contained an indication of whether the content of the next location was also to be printed. By this means equivalents of any desired length could be printed.

Some Characteristics of the Program

This program had two important features.

Firstly, all operations within the program were carried out on the first equivalents. As these were uniformly constructed, a greater simplicity was achieved than if the foreign language words or target language words had been processed directly.

Secondly, the distinct parts of the whole program were isolated, the linkages being supplied by the addresses in the first equivalents. Thus extra subroutines could be constructed and linked to the program merely by altering addresses in the relevant first equivalents. For instance, if a more refined condition routine was necessary for a certain set of first equivalents, this routine could be placed in the store and the second addresses of the first equivalents altered to the address of the initial order of the new routine.

The size of storage in the computer imposed severe limits on the extent and performance of the program. Thus very small dictionaries were used, although best use was made of the space available by means of stem-ending splitting. Apart from these faults, there were two inherent drawbacks of the above type of program.

The use of separate condition routines employing a matching procedure to examine the minor context of a first equivalent lead to an excessive program. A more economical approach would be to calculate correct alternatives from code numbers by some means. This would greatly reduce the storage space assigned to this particular part of the program.

Secondly, the method of effecting change of word order appears to be applicable only to subsections of languages where permutation of target language order into foreign language order is purely local. Thus if a set of n consecutive code numbers in S_2 was matched by the above method to a dictionary of structures, the change of word order was confined to the corresponding set of n first equivalents only. This process was clearly incapable of dealing directly with rearrangements of blocks of words. A possible solution of the problem here would be to use two structure-dictionaries, one for permuting elements within a block, another to permute the blocks. The necessity of using a structure-dictionary will disappear when a suitable technique of calculation (as opposed to matching) has been discovered.