[*Mechanical Translation*, Vol.6, November 1961]

# A Formula Finder for the Automatic Synthesis of Translation Algorithms

by Vincent E. Giuliano*, Computation Laboratory of Harvard University

A *system of procedures and computer programs is proposed for the semi-automatic synthesis of Russian-English translation algorithms.*

*For the purposes of automatic formula finding, a large corpus of Russian scientific and technical text may be processed by an automatic Russian-English dictionary, the resulting word-by-word translation post-edited according to a systematic procedure, and the final translation transcribed back onto magnetic tape for input to a computer. The operation of the proposed system is based on the automatic comparison of magnetic tapes containing the original automatic dictionary outputs with ones containing the parallel post-edited texts. It is expected that, when given proper clues, the formula finder will be capable of synthesizing algorithms that can be used to convert one text into the other.*

*The clues corresponding to a desired algorithm consist mainly of a list of logical variables that might in some combination govern the application of a specified post-editing transformation. Whenever a product of the transformation is found in the post-edited text, the formula finder examines the truth value configuration of the given variables in the automatic dictionary output. After examining all instances of the transformation, the formula finder ascertains whether the given variables can be combined into a logical formula that implies the given transformation. The formula finder compounds the given variables into a valid and optimal translation algorithm if it is at all possible to do so.*

The automatic production of accurate and reliable sentence-by-sentence translations between pairs of natural languages must await the resolution of complex syntactic and semantic problems whose solutions must ultimately be expressed as machinable algorithms. These are well-defined rules that operate on automatically interpretable information units. The central goal of much current research in the field of automatic language translation is to find and test such algorithms. For example, certain syntactic algorithms presently being studied at the Harvard Computation Laboratory are designed to remove many of the ambiguities of case and tense residual from word-by-word analyses of Russian. These particular algorithms reflect the governing influences of certain types of words upon their close neighbors, and hopefully will clear the way for the discovery of more sophisticated procedures to deal with larger phrases and clauses.

Problems of testing translation algorithms by machine have been discussed elsewhere and automatic programming systems are being devised to facilitate the communication of algorithms from man to machine.[1,2] The present paper is concerned with the semiautomatic synthesis of translation algorithms from empirical data, a means of formula finding that might eventually supplement current research methods. The proposed *formula finder* is a system of computer programs that will compare an extensive body of Russian text with its parallel English translation. When given proper clues by linguists, the programs will synthesize algorithms that can be used to transform one text into the other.

The formula finder system to be discussed here is compatible with the translating programs operating at Harvard.[3,4,5] Russian is therefore taken as the source language for translation, English as the target language. Nevertheless, the logical principles used in designing the formula finder are not language-dependent. These principles could be employed in the design of similar formula finders capable of operating with other given pairs of mutually translatable natural or artificial languages.

While an automatic formula finder may eventually serve as an important aid for research in automatic language translation, such a system cannot replace the linguists and other scholars currently engaged in this activity. The algorithms synthesized by the proposed formula finder are guaranteed to work only on the experimental corpus of text examined by the machine; they will be only approximately valid when applied to other texts. The synthesized algorithms must

* Now at Arthur D. Little, Inc. This work was supported by the National Science Foundation through a grant to Harvard University. The writer acknowledges the close collaboration of Professor Anthony Oettinger and of his other colleagues at the Harvard Computation Laboratory.

be examined, evaluated, and perhaps revised or generalized in the light of long experience with the languages by monitoring human linguists.

## 1. Translation Transformations

It is convenient to introduce a few symbolic conventions. The sentences in a corpus of Russian text to be translated or analyzed will be thought of as serially-numbered, the symbol $s_j$ being used to denote the $j$th sentence. Words, punctuation marks, special symbols, and other components of sentences will also be thought of as being numbered within their sentences, and the symbol $w_{ij}$ will be used to identify the $i$th component of the $j$th sentence.

An essential subsystem of the formula finder is an automatic Russian-English dictionary operating on inflected Russian word forms, i.e., a so-called "full paradigm" dictionary.[4] Although the Harvard dictionary contains Russian word stems, transformations of its outputs are provided that make the dictionary behave as if its words were represented by their full paradigms.[5] The transformation $T_d$ performed by the automatic dictionary replaces each Russian word $w_{ij}$ with an entire dictionary entry $W_{ij}$ for that word on magnetic tape, i.e., Wij = $T_d(w_{ij})$. When $w_{ij}$ is a punctuation mark or special symbol, $T_d$ replaces the symbol with a "dummy" dictionary entry $W_{ij}$ containing only that symbol and an appropriate amount of fill. Each regular dictionary entry is presumed to contain a Russian word, a complete set of English correspondents for that word, and coded grammatical data characterizing the Russian word and its correspondents in detail. Entries from the Harvard Automatic Dictionary, printed from magnetic tape, are shown in Fig. 1. A typical Russian word is shown transliterated and marked $\alpha$, the English meanings are marked $\beta$, and the coded data are marked $\gamma$. Part of the coded data, for example, reads *ND11N100.* These characters convey the information that the word *притяжение* functions as a noun (N), that it is declinable (D), that it belongs to a certain subclass of inanimate nouns *(II),* that it is neuter (N), that it functions in the singular only (1), and that it has no special forms (00). The other code characters, *N 10.00, A0*, and *A1*, indicate other pertinent properties of the word.[6]

The word-by-word transformation of the automatic dictionary $T_d$ induces a transformation on the sentences. Each sentence $s_j$ is replaced by a set of concatenated dictionary entries $S_j = W_{ij}$ called an *augmented sentence.* The basic research output of an

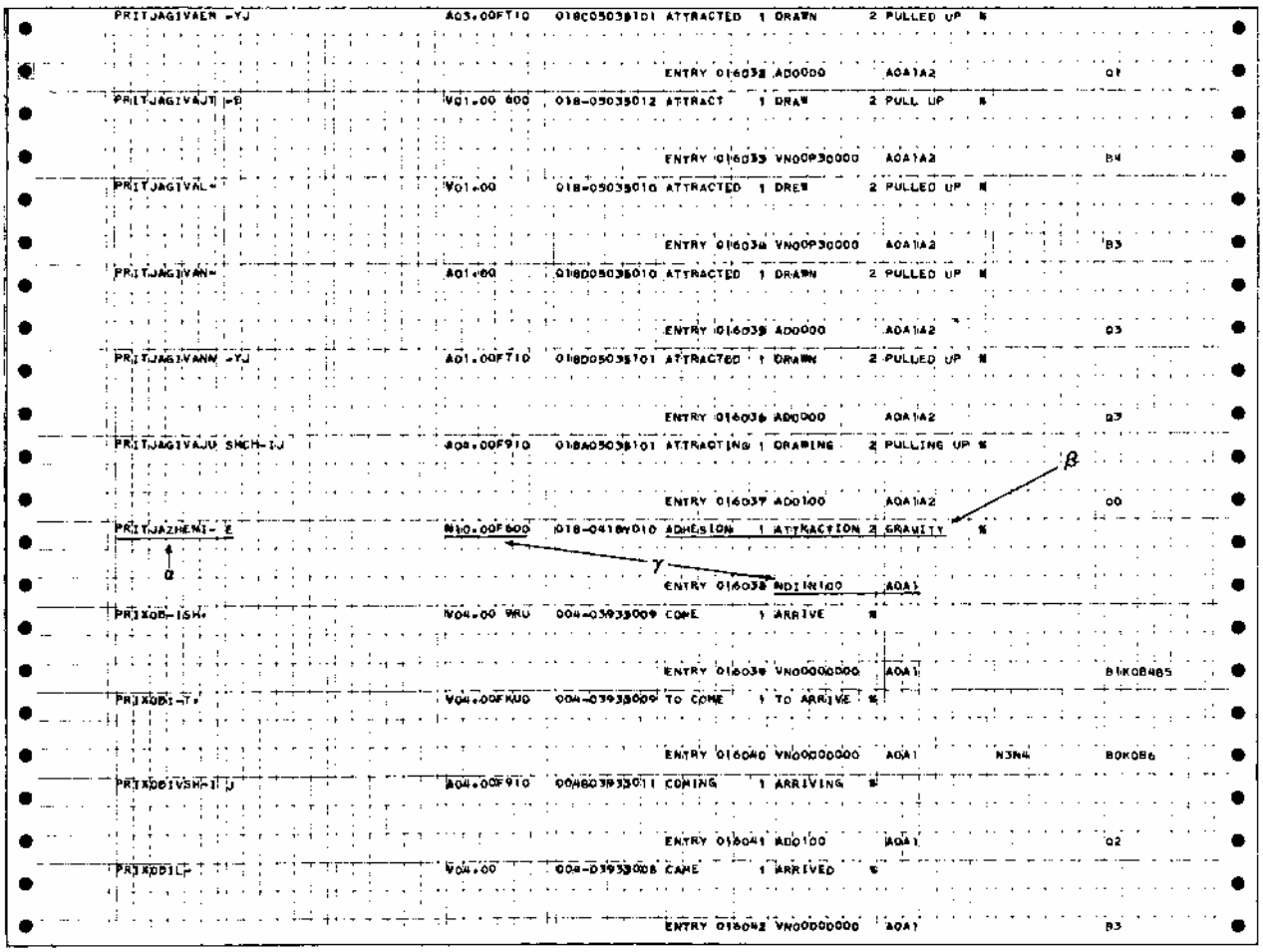



FIGURE 1
ENTRIES IN THE HARVARD AUTOMATIC DICTIONARY

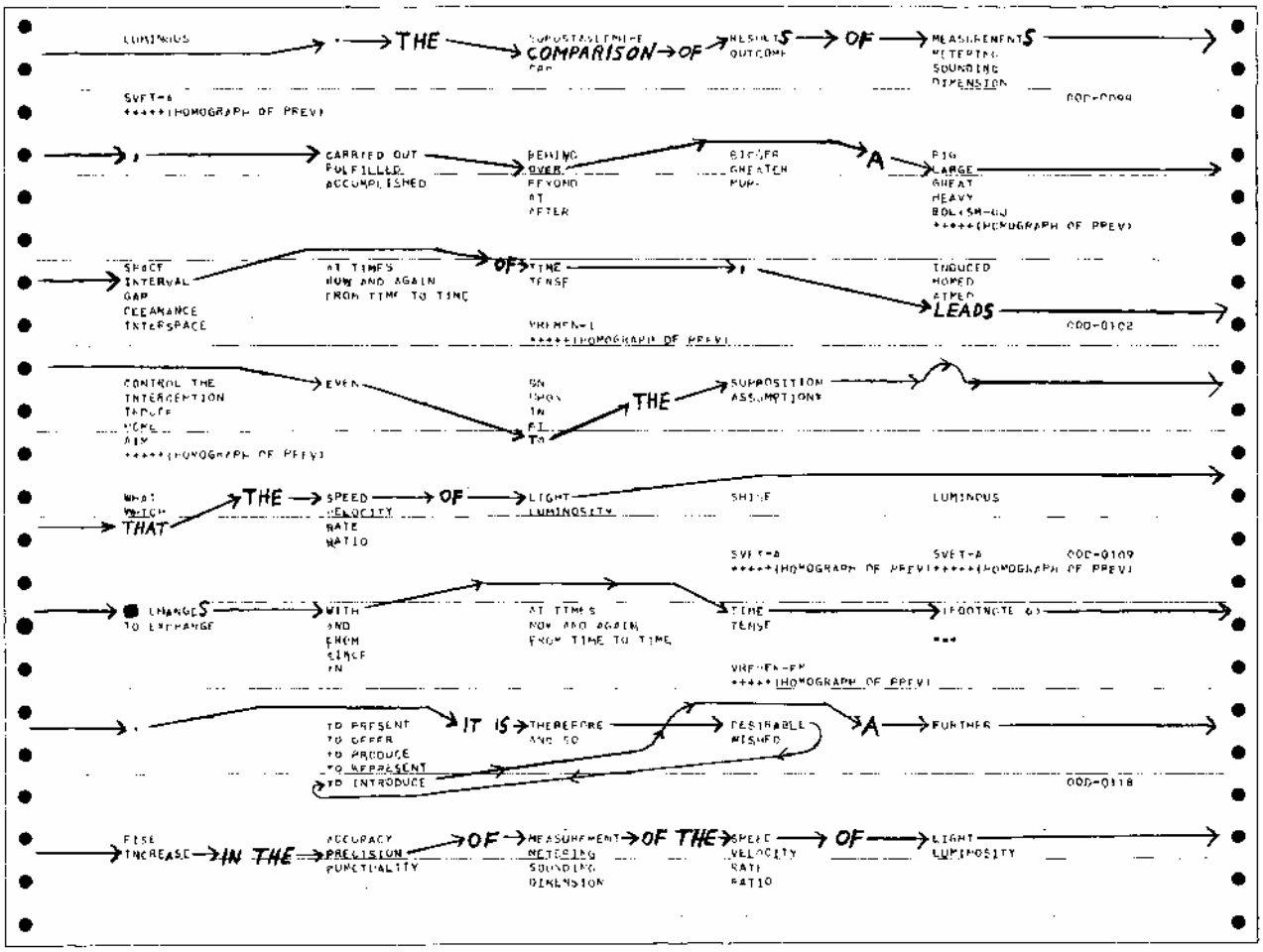



FIGURE 2
MACHINE-PRODUCED WORD-BY-WORD TRANSLATION, AFTER POST-EDITING

automatic dictionary is the set of augmented sentences $S_1,S_2\ S_3,\ .\ .\ .,S_p$ recorded on magnetic tape. This output will be called the *augmented text* for the given corpus. (In earlier publications, it has sometimes been referred to as the *text-ordered sub-dictionary.*[4]) The augmented text contains both the original textual data and the additional lexical data present in the dictionary. It is the logical input to any further automatic process that improves the translation by performing syntactic or semantic transformations.

Word-by-word translations produced by an automatic dictionary can be converted into smooth and idiomatic translations by post-editors familiar with the technical field of the material translated and having a slight knowledge of Russian.[4,5] A post-edited section of a word-by-word translation is shown in Fig. 2. The print is produced by a machine program that edits the data in an augmented text into a readable format. The post-editor has drawn arrows on the machine-produced print indicating a choice of English correspondents and word order. He has also inserted some short English words and has indicated other modifications in the printed text. At the date of this writing, about 40,000 running words of Russian text have been translated with the Harvard dictionary and post-edited in this manner.

The post-editor effectively behaves like a classical "black box" of electrical circuit theory. He determines a syntactic and semantic transformation $T_s$ that carries the word-by-word translation into a smooth and idiomatic translation. Although the output of this transformation can be measured for various values of the input, the internal operation of the post-editor cannot be viewed. While the post-editor may produce perfect copy, it does not necessarily follow that he, or anyone else for that matter, completely understands the process used in translating.

The operation of the formula finder is based on the machine comparison of augmented texts produced by an automatic dictionary and post-edited translations of the same texts. The post-edited translation of each $S_j$ will be represented by $E_j = T_s(S_j) = T_sT_d(w_{ij})$ where $T_d$ is the automatic dictionary transformation, and $T_s$ is the transformation determined by the post-editor. The formula finder simultaneously examines each $S_j$ and its corresponding $Ej$. It establishes correspondences between the parallel texts and synthesizes

algorithms defining portions of $T_s$ valid for the experimental corpus.

The transformation $T_s$ defines only one of the many possible mappings of the given Russian corpus into a valid translation, namely, that actually used by the post-editors. Use of other post-editors, or even the same post-editors at different times, would result in somewhat different definitions of $T_s$. The non-uniqueness of the post-editing transformation need not be a serious problem at present, however, provided that steps are taken to insure the self-consistency of $T_s$. At this stage of research, what is desired is a single valid system of rules for translating, not a catalogue of rules for obtaining all alternative valid translations. Emphasis is therefore to be placed on the use of a fixed set of post-editing conventions designed to lead to as simple and self-consistent a definition of $T_s$ as possible.

## 2. Translation Algorithms

A tabular definition of $T_s$ is provided by the list of $S_j$ and corresponding $E_j$. This definition amounts essentially to a dictionary of sentences in the experimental corpus and their translations into English. Since it is obviously not possible to store or even to generate all meaningful Russian sentences, this definition is not useful when it comes to translating other Russian texts. What is needed is a factorization of $T_s$ into a product of machinable algorithms applicable to situations commonly occurring within sentences. For purposes of automatic formula finding, a specific type of factorization is assumed:

$$T_s = A_1A_2A_3A_4...A_n \qquad (1)$$

where the $A_r$ are elementary transformations having the $W_{ij}$ as their arguments; they are called *basic algorithms.*

### A. THE LOGICAL STRUCTURE OF BASIC ALGORITHMS

The basic algorithms to be derived by the formula finder are presumed to have a certain logical structure, the motivation for which has been given elsewhere.[2,4] It must be possible to state each $A_r$ algorithm in a form similar to that of a logical implication:

$$D_r:W_r \rightarrow B_r \qquad (2)$$

where $D_r$ and $W_r$ are open sentences* stated in the language of a first order logical calculus, and $B_r$ is an editing action. When translating by machine, the action $B_r$ is to be taken in textual contexts where logical propositions corresponding to $D_r$ and $W_r$ are both true. The distinction between $D_r$, called the *determiner formula,* and $W_r$, called the *working formula,* is treated in Ref. 2. Roughly speaking, $D_r$ states the general condition for applicability of a given algorithm (for example, the presence of a genitive noun), while $W_r$ contains the detailed logic of the algorithm. Both $D_r$ and $W_r$ are compounded out of certain admissible predicates and the usual connective functors of the propositional calculus: • for *and,* $\vee$ for or, and $\sim$ for *not.*

The predicates used in the $D_r$ and $W_r$ formulas must be functions of the $W_{ij}$. A typical predicate might, for example, correspond to the statement: "$w_{ij}$ is a verb." At a given position in an augmented Russian text, the values of $i$ and $j$ are fixed numbers and the predicates correspond to propositions that are either true or false. In other words, textual position serves as a basis for quantifying the $i$ and $j$ variables in open sentences while translating. It is sometimes convenient to use a single name to denote either a predicate or any of the propositions associated with that predicate for specific values of $i$ and $j$. Accordingly, the term *variable* will be used to denote either a predicate or any of the binary valued propositions obtainable from it by assigning particular values to $i$ and $j$. Variables will be represented by the symbols $\phi_1, \phi_2, \phi_3, \ ... \ \phi_n$, etc. The *specification* of an admissible variable at a given text position is the truth value of the proposition. Only variables that can be specified automatically are admissible; the automatic specification of variables is discussed in part 4 of this paper.

At each contextual position, $D_r$ and $W_r$ become closed sentences that are either true or false. The truth values of the closed sentences are determined by the specifications of the component variables. The truth value associated with a given formula in a given context will be called the *evaluation* of the formula for that context.

From the viewpoint of automatic formula finding and testing, it is desirable to search for algorithms that are free of interaction, algorithms that can be derived and studied in isolation from one another. A sufficient condition for the independence of two algorithms $A_r$ and $A_r$ , is that they commute, i.e., that $A_rA_r$, $S_j = A_r,A_rS_j$ for every $S_j$. It is possible to give examples of noncommuting basic algorithms, in particular, algorithms involving permutations of word order. For example, suppose that the action transformation *E(i-1,i)* leads to the exchange of the translations of the (*i-l*)st and *i*th text words. The algorithms $D_r:W_r \rightarrow E(i\text{-}l,i)$ and $D_r:W_r \rightarrow E(i,i + l)$ obviously do not commute if there are values of $i$ and $j$ that make both $D_r$ and $W_r$ true propositions.

The problem of algorithm noncommutativity can be greatly alleviated by restricting the types of admissible modifications that can be made while post-editing. If the post-editing transformation $T_s$ is to be approximated by a product of commuting algorithms, then it must be kept as simple, straightforward, and self-consistent as possible. The post-editing instructions listed in Part 3 of this paper are framed with this objective in mind. In particular, word order interchanges are discouraged. Even assuming restrictions on $T_s$, however, it may still not be possible to express the complete transformation $T_s$ as a product of commuting basic algorithms. The primitive formula finder

* *Open sentences* are logical entities sometimes referred to in the literature as *statement matrices* or *propositional functions.* The usage followed here is that suggested by Quine in Ref. 7.

discussed here can synthesize only a single basic algorithm at a time. The validity of each derived algorithm will therefore depend to some extent on whether it is free of interaction with the others.

### B. A SAMPLE TRANSLATION ALGORITHM

Most of the syntactic and semantic algorithms proposed in the literature of machine translation can be stated as basic algorithms or as chains of basic algorithms. For example, a rule selected out of several given by Fargo and Rubin will be considered: [8,*]

"Rule number III—'Translation of genitive suffix'

1. Is *immediately* preceding item: *a noun without K, personal pron. or participle with a noun function?*
   (a) If yes, translate suffix by *of*
   (b) if no, see 2. . . ."

Predicates $\phi$, involved in the algorithm are:

| | |
|---|---|
| N(i) | $w_{ij}$ is a Russian noun |
| G(i) | $w_{ij}$ is in the genitive |
| "K"(i) | $w_{ij}$ is the Russian word "K" |
| PP(i) | $w_{ij}$ is a personal pronoun |
| PA(i) | $w_{ij}$ is a participle |
| NF(i) | $w_{ij}$ can function as a noun |

(3)

Since the same rule holds for all sentences, the index $j$ is suppressed in the symbolic names for the predicates. Information enabling the automatic specification of each of these variables is present in the form of grammatical codes in the entries of the Harvard Automatic Dictionary. The indicated action $B_r$ can also be assigned a symbolic name, *INS(xxx,i)* standing for *insert the string of characters xxx before the translation of* $w_{ij}$. When applying the rule to nouns, the determiner formula is *N(i)* • *G(i)*. The complete basic algorithm is:

$$\text{N(i)} \bullet \text{G(i)} : [\text{N(i-l)} \bullet \sim \text{"K"(i-2) VPP(i-l) VPA(i-l)} \bullet \text{NF(i-l)}] \rightarrow \text{INS(of,i)} \quad (4)$$

### C. TRIAL TRANSLATION AND FORMULA FINDING

The language of the logical calculus is simple and mnemonic, and appears to be well suited for the formulation of translation algorithms. A computer program that interprets formulas stated in this language is currently being used at Harvard as a tool for research on Russian syntax. The program goes through a large corpus of augmented text and selects all word contexts that satisfy a given formula. The contexts are then automatically edited, printed, and studied by linguists.† A design for a more advanced system that uses the language of basic algorithms, called *trial translator,* has been proposed elsewhere.[2] The trial translator applies experimental basic algorithms to augmented texts in order to produce improved translations. Its operation is based on the automatic association of basic algorithms with dictionary entries, the automatic specification of variables, and the automatic evaluation of formulas.

The proposed formula finder and trial translator systems are compatible; the former enables the semiautomatic derivation of basic algorithms, the latter enables the automatic testing of such algorithms. When a linguist wishes to derive an algorithm, he furnishes the formula finder with a definition of the action $B_r$ that he wishes to study, a determiner formula $D_r$ for that action, and a list of variables $\phi_1, \phi_2, \ldots \phi_n$ that he feels might be of importance in determining that action. The formula finder compounds the given variables into a working formula $W_r$ if it is at all possible to do so, thus defining a complete basic algorithm $D_r{:}W_r \rightarrow B_r$. The basic algorithm is produced in both a readable format for human inspection and a machinable format for input to the trial translator. Information feedback relationships will exist between the formula finder system, the trial translator system, and the monitoring human linguists; these are discussed in Part 5 of this paper.

The power of an algorithm synthesized by the formula finder will depend on whether the most important lexical variables are included in the list $\phi_1, \phi_2, \ldots, \phi_n$. A derived working formula, when taken together with the given $D_r$, will always describe sufficient conditions for executing the given action $B_r$ in the experimental corpus. In some cases, however, a derived $W_r$ might describe both necessary and sufficient conditions for consummating the action $B_r$, given that $D_r$ is true. Algorithms containing such working formulas will be called *maximal* since they cannot be improved insofar as the experimental corpus is concerned. In trial translating, both maximal and nonmaximal algorithms can be used; a single action $B_r$, can occur in several algorithms having different determiner and working formulas.

## 3. The Preparation of Parallel Texts

The proposed formula finder system is block-diagrammed in Figs. 3 and 4. The process divides naturally into two parts. The first part, illustrated in Fig. 3, is concerned with the preparation of parallel texts; the second part is concerned with the machine derivation of basic algorithms (Fig. 4).

The grist from which the formula finder is to synthesize algorithms is a large and representative corpus of Russian technical text. This corpus must be processed by an automatic dictionary and be available in the form of augmented texts recorded on magnetic tape. Machine-printed word-by-word translations must also be prepared from the augmented texts and made available for post-editing. Since the derived formulas will be strictly valid only for the sentences in the given corpus, it is important that the corpus be as extensive and representative as possible. Initially, there might be advantages to covering one or two technical fields in depth, say electronics and instru-

* This algorithm is mentioned for illustrative purposes only; the present writer does not assert that it is necessarily valid. The expression *noun without K* will be taken to mean *noun not preceded by the Russian preposition K*, but the writer is not certain that this is the meaning intended by the authors of the algorithm.

† The context-selecting program was written by W. Bossert.

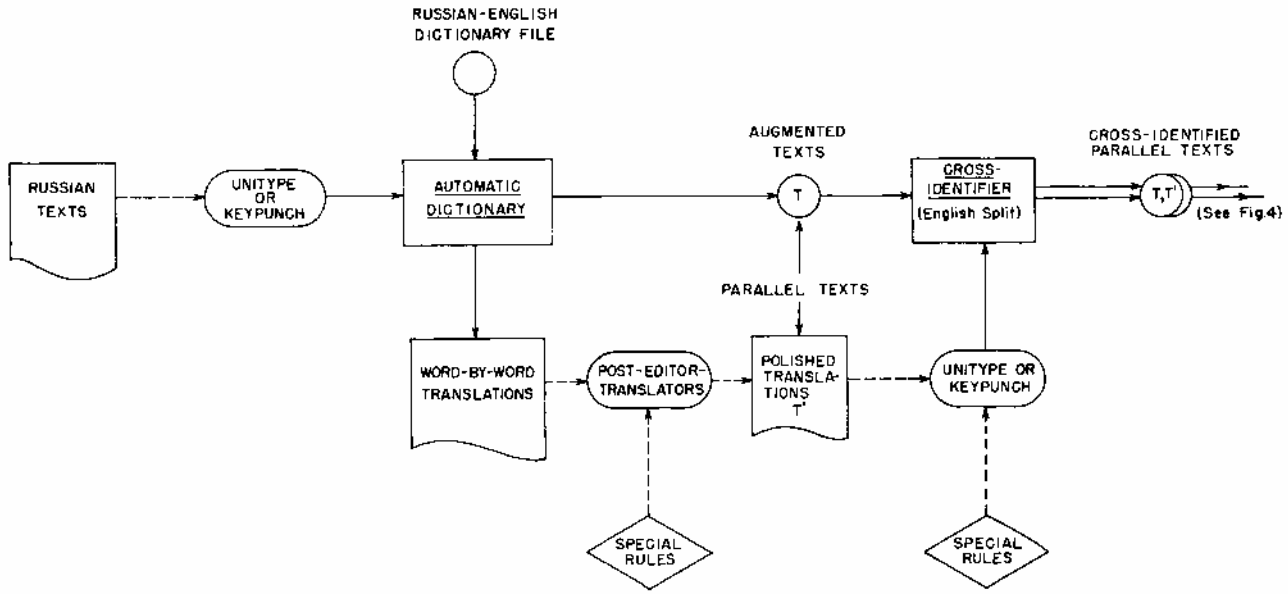



FIGURE 3
THE PREPARATION OF PARALLEL TEXTS

mentation, and excluding material from other fields. Later, after a certain number of fundamental algorithms have been found and tested, the corpus could be extended to cover other technical fields having their own particular idioms and constructions.

Our experience indicates that post-editing can readily be accomplished by drawing lines and entering information on machine-produced prints like that shown in Fig. 2. The information on a post-edited print can rapidly be transcribed into conventional

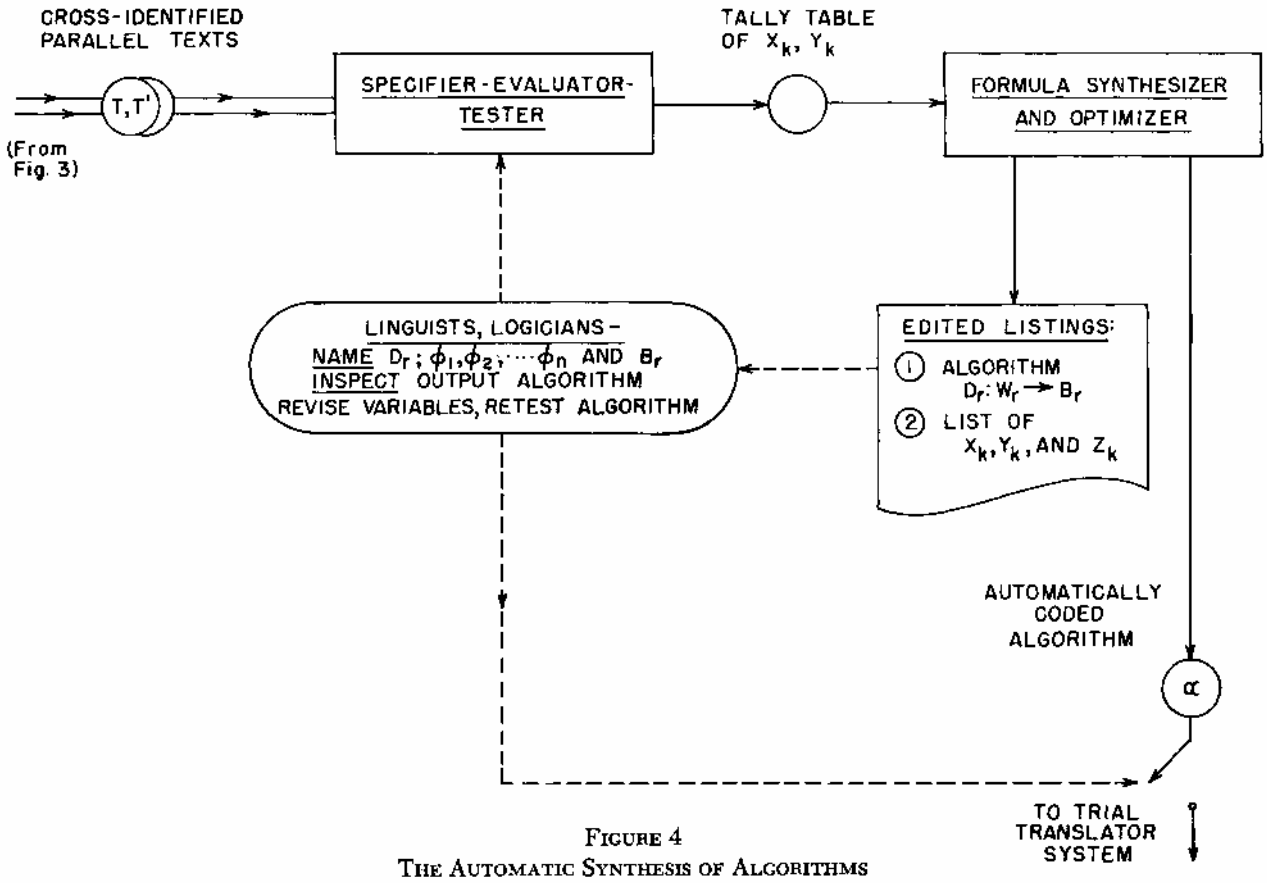



FIGURE 4
THE AUTOMATIC SYNTHESIS OF ALGORITHMS

running format by a typist who simply copies the words at the heads of arrows.

### A. POST-EDITING TEXTS

Post-editors must be confined to making transformations that are reasonably consistent and that can potentially be automatized through the use of commuting basic algorithms. Rules must therefore be provided that limit the scope of $T_s$. The formulation of a concise set of post-editing rules must await the detailed designing and programming of a working system. Nevertheless, it is possible to cite tentative rules that illustrate the types of transformation that can most probably be accommodated:

*Post-editing Rules Governing Text Transformations*

(1) The original Russian word order should be preserved whenever it is at all possible to do so and still obtain a clear translation, even when a loss of elegance results. For example, . . . *колебаний напряжения триггера* . . . should be translated ... *of the oscillations of the voltage of the trigger* . . . rather than by the smoother inverted construction . . . *of the oscillations of trigger voltage* ... .In any event, the translation should be no more sophisticated than a sentence-by-sentence translation. The translations of words can be moved about within a sentence when this is absolutely necessary, but they must never be moved from one sentence to another. Naturally, the sequence of sentences must also be preserved.

(2) Normally, the English words used in the post-edited text should be selected from the correspondents printed in the word-by-word translation or from a special list of short particle words. The list of particles is treated in post-editing rule *(4)*. Printed correspondents may be modified according to rule *(5)*. Now and then it may not be possible to translate a Russian word correctly using the printed English correspondents, or the word might be missing from the dictionary and shown transliterated instead of translated. When such is the case, the correct English correspondent should be written directly under the existing English correspondents, if any, for the word concerned.

(3) Any word can be given a null translation; i.e., no translation of it need appear in the post-edited copy.

(4) Certain special short words, given on a list furnished to the post-editor, can be inserted as needed in the post-edited translation. Among the words on this list are:

(a) Forms of the verb *to be,*
(b) Articles such as *the, a,* and *an,*
(c) English prepositions sometimes rendered in Russian by case endings, for example, *to, of, for, by,* etc.

(5) The form of a printed English correspondent can be modified so that it correctly represents the proper number, person, mood, tense, etc. For example, *s,* or *es* can be added to a noun form to make it plural, *ing* might be added to a verb in order to generate a participle, etc.

(6) Commas, colons, and semicolons can be inserted or deleted when an absolute necessity for such a change exists, but the original sentence structure should be retained insofar as this is possible.

(7) In some cases, it may be possible to translate a passage only awkwardly if rules *(1)-(6)* are followed. If an awkward translation made according to the rules is nevertheless accurate and understandable, it should be retained in the post-edited copy. The post-editor has the option of following such an awkward passage with a superior handwritten translation made in violation of rules *(1)-(6),* provided that the improved version of the passage is enclosed within special symbols, say dollar signs, for later machine identification.

(8) In some cases, it may be absolutely necessary to violate one of the rules *(1)-(6)* in order to translate a word, phrase or sentence adequately. In such cases the rules can be violated, but the affected portions of the text must be surrounded by special symbols, say asterisks.

Rules *(7)* and *(8)* provide means for preserving information that cannot initially be handled by the machine system. This information can be automatically retrieved for processing at a later date. These two rules also allow scholars and translators who take pride in their work to complete usable translations without doing violence to their aesthetic senses. The post-edited translations should be of sufficiently high quality so that only a small additional amount of editing is required to prepare them for publication.

The text sample of Fig. 2 was post-edited according to the rules just enumerated. The post-editor has made a change in word order according to rule *(1)*, added new English correspondents according to rule *(2)*, deleted the translations of homographic Russian words according to rule *(3)*, inserted short words according to rule *(4),* altered existing correspondents according to rule *(5)* and deleted a comma according to rule *(6)*. It was not necessary to resort to the escape provisions of rules *(7)* or *(8)*. The transcribed passage reads fairly smoothly:

> The comparison of results of measurements, carried out over a large interval of time, leads even to the supposition that the speed of light changes with time (footnote 6). It is therefore desirable to introduce a further increase in the precision of measurement of the speed of light . . .

For the purpose of simplifying $T_s$ and thus facilitating speedy convergence to a valid set of algorithms, it may be desirable to adopt even more restrictive post-editing rules than those already suggested. These rules could even go so far as to require the uniform treatment of certain specific grammatical situations. Problems of systematizing the post-editing process have been discussed elsewhere, and specific procedures designed to insure a maximum degree of consistency have been suggested.[9] Initial experiments in automatic formula finding might well be based on the use of a relatively small text corpus that has been systematically post-edited according to such a rigid set of rules.

### B. THE TRANSCRIPTION OF POST-EDITED TEXTS

A strict word-by-word cross-identification between the transcribed post-edited text and the augmented text is required for the operation of the formula finder.

That is, the machine must be able unambiguously to identify the individual English words in the post-edited text with the $W_{ij}$ entries in the augmented text. The necessary cross-identification can be effected automatically, but only if some additional information relating to word order changes is supplied to the machine. This information can be supplied by the typist who transcribes the post-edited text back onto magnetic tape, and can be encoded along with the text itself. The coding scheme should enable resolution of all ambiguities due to skipped words and changes in word order, but yet should be as simple as possible. The typist might, for example, be directed to observe the following instructions for transcribing and encoding texts:

*Instructions for Transcribing Post-edited Texts onto Magnetic Tape*

(1) *Explanation of Format.* Machine printing appears in five fixed positions across each line of text; each of these positions holds an *entry.* An entry may contain several English correspondents arranged in a column, a punctuation mark, or a comment. An English correspondent written by a post-editor directly under the machine printing for an entry is considered to be part of that entry. Short English words written in by a post-editor, such as *the, an, a,* etc., are considered to be *insertions;* they are not part of any entry.

(2) *Instructions.* Type the English words and punctuation marks at the heads of the arrows in a normal running format. The arrow will normally proceed from left to right across the page, selecting an English correspondent out of each entry. When the arrow skips forward over one or more entries or circles backwards, it is necessary to insert a *position number* in the text according to the following rule:

> When the arrow skips forward or circles backwards, insert in the corresponding position in the transcribed text a number prefixed by a plus or minus sign indicating the relative position of the next entry selected. The number must be surrounded by parentheses for machine identification. For example, if the arrow skips over two entries, the "(+3)" is to be inserted. The position number "(-2)" means *two entries back,* etc. Include any short insertion words in the transcribed copy, but do not count them in computing the position number.

If the convention for recording position numbers is followed in transcribing the sample post-edited text of Fig. 2, the following copy is obtained:

> "THE COMPARISON OF RESULTS OF MEASUREMENTS, CARRIED OUT OVER (+2) A LARGE INTERVAL (+2) OF TIME, LEADS (+2) EVEN TO THE SUPPOSITION (+2) THAT THE SPEED OF LIGHT (+3) CHANGES WITH (+2) TIME (FOOTNOTE 6). (+2) IT IS THEREFORE DESIRABLE (-2) TO INTRODUCE (+3) A FURTHER INCREASE IN THE PRECISION OF MEASUREMENT OF THE SPEED OF LIGHT . . ."

Since a word-by-word translation is simply a machine-edited version of an augmented text, the entries in the former are in one-to-one correspondence with those in the latter. The position numbers therefore define a precise correspondence between the words selected by post-editors and the associated entries in the augmented text.

### C. AUTOMATIC CROSS-IDENTIFICATION

The typist will make occasional mistakes while transcribing the large corpus of post-edited text onto magnetic tape. If position numbers are assigned incorrectly or if words are mistakenly left out or transposed, there will be "phase" errors in the encoded correspondence between the tape containing the post-edited text and that containing the augmented text. A machine program called *cross-identifier* is therefore included in the flow pattern of Fig. 3 to check the word-by-word association given by the position numbers. It verifies that the English correspondents used by the post-editors are, in the majority of cases, also contained in the associated $W_{ij}$ entries.

Automatic cross-identification is complicated by the fact that the forms of English words may be modified according to post-editing rule (5). Before English words in the post-edited text can be compared with words in the augmented text, they must all somehow be reduced to standard forms that can be matched automatically. This can be accomplished by automatically removing standard inflectional endings, like *s, es, ing,* etc., from English word forms, thereby reducing the inflected word forms to more or less standard stem forms.

Machinable rules for the automatic splitting of word affixes, a process sometimes called "inverse inflection," have been developed for Russian, a language that has a much more complicated system of suffixes than English.[10,11] The development of similar rules for the automatic inverse inflection of English words should pose no fundamental linguistic problems. Research in this direction is presently underway at the Harvard Computation Laboratory. The projected cross-identifier program will incorporate the necessary rules for separating English stems and affixes. Each English word in both the post-edited text and the augmented text will be automatically split into a stem and an affix. The cross-identifier will then compare only stems; each stem in the post-edited text will be matched against the stems originating from the corresponding $W_{ij}$ entry. The reduction of words to stems will thus enable an automatic check on the typist's position number coding, even when English forms are modified according to post-editing rule (5).

The list in "insertion" words, *a to, of,* etc., is to be carried in machine memory during the cross-identification process. The cross-identifier program will recognize these words as exceptions, and will not attempt to locate them in the $W_{ij}$ entries. The machine can therefore always check the word-entry association encoded by the typist except when a new English meaning is assigned to an existing entry.

When the cross-identifier finds an isolated word in the post-edited text that is not in the corresponding $W_{ij}$ entry, it assumes that the word is a new one as-

signed according to post-editing rule *(2)*, and that the association encoded by the typist is correct. When several running words are found that cannot be matched with the corresponding $W_{ij}$ entries, the cross-identifier assumes that a phase error or unusual idiomatic construction is present. The affected sentence is deleted from the experimental corpus and recorded on a separate tape, and the machine proceeds to the next sentence. Since post-editing is always done on a sentence-by-sentence basis according to rule *(1),* errors in identification will always be localized. The cross-identifier will also delete portions of the translation made in violation of post-editing rules *(l)-(6)* and enclosed in dollar signs or asterisks, and record them on another separate tape. The separate tapes can eventually be printed and the problematic sentences subjected to further study.

The result of cross-identification is a table of correspondences between the individual words in the post-edited text and the $W_{ij}$ entries in the augmented text. This tabular correspondence might be automatically encoded by inserting appropriate markers into the $W_{ij}$ entries themselves. The table provides a word-by-word definition of the transformation $T_s$. This is a more finely structured definition of $T_s$ than the list of corresponding $S_j$ and $E_j$ , but is still not one that can be practically used for translating other texts. The second portion of the formula finder system, block-diagrammed in Fig. 4, is concerned with deriving the $A_r$, the basic algorithms in the assumed decomposition of $T_s$.

## 4. The Synthesis of Basic Algorithms

Parallel texts need be prepared only once by the process of Fig. 3; thereafter they can be used for the derivation of any number of basic algorithms. The synthesis of each algorithm requires a separate iteration of the process diagrammed in Fig. 4. Prior to a given algorithm-synthesizing run, a linguist must furnish the computer the following clues concerning the desired algorithm:

(1) A definition of $B_r$, the action portion of the desired algorithm. In the sample algorithm, the action was *INS (of, i);* other typical actions might relate to the selection of a particular English correspondent, the inflection of a correspondent into the plural, etc.[2]

(2) A determiner formula $D_r$ for the desired algorithm. This is the portion of the algorithm known beforehand; it limits the machine to investigating textual situations known to be pertinent. The determiner *N (i)* • *G(i)* given in the sample algorithm would limit the formula finder to investigating the insertion of *of* before genitive nouns, and a derived algorithm would not be complicated by other *of* occurrences.

(3) A set of predicate "variables" $\phi_1, \phi_2, \ldots \phi_n$ having the $W_{ij}$ as their arguments. They are, in the opinion of the monitoring linguist, the building blocks of a potential working formula $W_r$. The list may include many more variables than will actually be needed in the formula; the machine will use only those variables that are actually required.

### A. THE AUTOMATIC SPECIFICATION OF VARIABLES AND EVALUATION OF FORMULAS

Variables in the determiner formula and in the set $\phi_1, \phi_2, \ldots \phi_n$ must be admissible, i.e., provisions must exist for automatically specifying their truth values in all textual instances. Only variables which relate to the morphology of Russian or English words or to lexical data present in the $W_{ij}$ entries of an augmented text can be specified automatically.

Certain predicate variables can be specified by means of the comparison of a known string of characters, given by the variable, with other strings of characters in the $W_{ij}$ entries. Such predicate functions will be called *string variables.* In the Harvard dictionary, for example, entries contain coded "part of speech" markers, *N, A,* etc. (standing for *noun, adjective,* etc.) in a fixed field, character position *313*. In order to specify *N(i+2),* then, it is sufficient to investigate character position *313* in the second entry following that under principal consideration. If the character in this position is *N,* the specification is *1* (true), otherwise the specification is *0* (false). The "part of speech" variables, then, are string variables, as are indeed all the variables in the sample list *(3)*. *Since string variables deal directly with the available lexical and morphological units, it is possible to formulate any admissible basic algorithm in terms of them.*

A relatively simple computer routine can be designed for the automatic specification of string type variables. Indeed, the presently operating context selecting program incorporates a specifier routine capable of handling monadic string variables like those in the sample list *(3)*. A more powerful string-variable specifier routine, capable of handling relational variables and variables with special quantifiers, is a required component of both the trial translator and formula finder systems.[2,12] Admissible string variables are those that can be defined by coded expressions which this routine is capable of interpreting. For example, the coded expressions for a monadic variable might contain:

(1) *A key.* This is a string of one or more known alphanumeric characters. The characters might represent part or all of a Russian or English word, or a grammatical code marker.

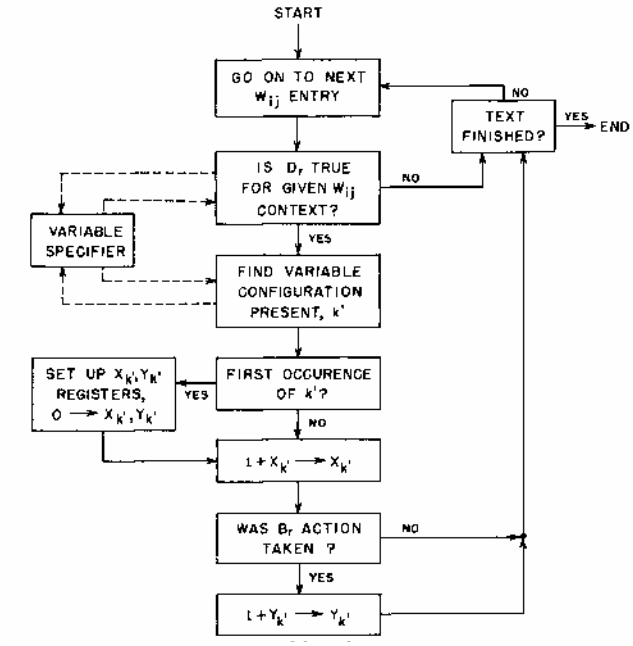
(2) *A major coordinate.* This specifies the entries in which search is to be made. The major coordinate is a relative coordinate, and is *0* for the augmented text entry under principal consideration, *-1* for the preceding entry, *+1* for the following entry, etc. The major coordinate may denote either a fixed entry or a set of entries that must be searched. Search might be made, for example, in all entries following the entry under primary consideration but preceding the next period. Provisions should be made for both backward and forward search, with limits determined by a secondary key.

(3) *A minor coordinate.* This specifies the location or locations within an entry that must be checked by

the specifier. It can be a number which denotes a specific field within an entry. In the Harvard Automatic Dictionary, for example, English correspondents, Russian stems, and coded grammatical data, with minor exceptions, occupy fixed fields. The minor coordinate might instead denote character positions that are search limits within an entry. The string in the sample propositional function *N(i+2)* is *N*; the major coordinate is *+2*, the minor coordinate is *313*.

When a monadic string variable is being specified, the program searches the data positions in the $W_{ij}$ entries defined by the major and minor coordinates. The strings thus obtained are compared with the key string. When a search is successful, the specification of the variable is *1*, otherwise it is *0*. Specifier-code expressions can also be used to define relational variables. For example, a dyadic variable can be defined by two keys, the corresponding major and minor coordinates, and an indication of the relation involved. Linguists should be encouraged to name variables mnemonically, for example, by writing *A(i), ADJ(i), ADJECTIVE*(i), etc. Such mnemonic names need be converted into specifier-code instructions only once, by a programmer, and the correspondence retained in an automatically-readable cross-reference table. The conversion of variables from mnemonic to specifier-code form can thereafter be done automatically.

A string variable specifier program is a component of the *specifier-evaluator-tester* program shown in the diagram of Fig. 4. Special specifier subroutines might also be included in this program for economically specifying predicate functions more complicated than string variables. The specifier-evaluator-tester program must also contain provisions for the automatic truth-value evaluation of determiner formulas. In a given context, the evaluation of a logical formula is




BASIC FLOWCHART FOR SPECIFIER-EVALUATOR-TESTER

determined by the specifications of the variables contained in that formula. There are several well known methods for evaluating logical formulas, any one of which can readily be programmed.[12,13,14] Our experience at Harvard indicates that a particularly simple evaluation process can be used if a formula is stated in disjunctive normal form, as a sum ($\vee$) of products ($\bullet$) in which only single variables are negated. An evaluator program now operating at Harvard requires only about a hundred lines of Univac coding.[12]

Besides provisions for the automatic specification of variables and evaluation of formulas, the specifier-evaluator-tester must also incorporate a simple subroutine capable of verifying whether the action $B_r$ has been taken at any given position in the post-edited text. This routine should be capable, for example, of determining whether *of* is inserted at any given position. It is in essence another specifier routine, one that operates on the post-edited text. It will be called the *action tester*.

### B. THE OPERATION OF THE SPECIFIER-EVALUATOR-TESTER

The inputs to each run of the specifier-evaluator-tester are the cross-identified parallel texts and a particular set $\{D_r; B_r; \phi_1, \phi_2, \ldots \phi_n$. A skeletal flow chart of the program is given in Fig. 5. The program simultaneously advances the two tapes containing the parallel texts; the cross-identification codes are used to keep the tapes in phase. As each new $W_{ij}$ entry is encountered, the program specifies the truth values of the variables in $D_r$ for the given values of $i$ and $j$. The program then evaluates the truth value of $D_r$ in terms of the truth values of the component propositions. When $D_r$ is not true, no further action is taken in the given context; the parallel texts are advanced (within a given sentence $i+1$ replaces $i$), and $D_r$ is evaluated for the next $W_{ij}$. When $D_r$ is true the program executes certain specifying, testing and incrementing operations before proceeding to the next item. These operations will be described, but first a brief paragraph will be devoted to a review of a topic of elementary logic, truth value configurations.[7,14,15]

There are $2^n$ possible configurations of truth values of the variables $\phi_1, \phi_2, \ldots \phi_n$ ; these correspond to the rows in the schematic listing of Table 1. A *1* in any position is here taken to mean that the corresponding $\phi_v$ is true in the given configuration, a *0* that it is false. Thus, in the first configuration all the $\phi_v$ are false; in he last all the $\phi_v$ are true. The configurations are uniquely identified by the binary patterns of the *1*'s and *0*'s; each row in the configuration table corresponds to a binary number $k$ between 0 and $2^n$—1. The number $k$ can therefore be used as a name for the corresponding configuration of variables.

Two sets of index registers, $\{X_k\}$ and $\{Y_k\}$, are set up and retained within machine memory during the specifier-evaluator-tester run. The values of $k$ correspond to the configurations of the $\phi$, that are actually

TABLE 1

CONFIGURATIONS OF LOGICAL VARIABLES

| k | $\phi_1$, | $\phi_2$, | $\phi_3$, | ..., | $\phi_{n-2}$, | $\phi_{n-1}$, | $\phi_n$ | Interpretation |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | All $\phi_v$ are false. |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Only $\phi_n$ is true. |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | | Only $\phi_{n-1}$ is true. |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | | |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | | |
| 5 | 0 | 0 | 0 | 1 | 0 | 1 | | |
| . | . | . | . | . | . | . | . | |
| . | . | . | . | . | . | . | . | |
| . | . | . | . | . | . | . | . | |
| $2^n$-5 | 1 | 1 | 1 | 0 | 1 | 1 | | |
| $2^n$-4 | 1 | 1 | 1 | 1 | 0 | 0 | | |
| $2^n$-3 | 1 | 1 | 1 | 1 | 0 | 1 | | All $\phi_v$ are true except $\phi_{n-1}$, |
| $2^n$-2 | 1 | 1 | 1 | 1 | 1 | 0 | | All $\phi_v$ are true except $\phi_n$. |
| $2^n$-1 | 1 | 1 | 1 | 1 | 1 | 1 | | All $\phi_v$ are true. |

encountered in the text corpus for contexts that make $D_r$ true. When $D_r$ *is* true, the variable specifier routines are used to determine the truth values of each of the $\phi_1$, $\phi_2$, ... $\phi_n$. The pattern of *1*'s (trues) and *0*'s (falses) thus obtained defines a logical configuration $k'$ that characterizes the state of the $\phi_v$ variables at the given textual position. When a given configuration $k'$ is thus encountered for the first time in the corpus, the machine sets aside two index registers, one for $X_{k'}$ and one for $Y_{k'}$, the numbers in both registers being initially set to zero. Then, and whenever the same $k'$ configuration is encountered in subsequent contexts for which $D_r = 1$, the specifier-evaluator-tester increments the number in the $X_{k'}$ register by 1.

After an $X_{k'}$ register is incremented, the action-testing routine is called into play. It ascertains whether or not the post-editor has taken the given action $B_r$ in the post-edited text. The position to be tested in the post-edited text is that corresponding to the context for which $D_r = 1$; it is located by use of the cross-identification table. If no action was taken, the program simply goes on to the next $W_{ij}$ entry, evaluating $D_r$ again, etc. If the action $B_r$ was indeed taken, the program increments the $Y_{k'}$ register by 1 before going on to the next item. The specifier-evaluator-tester program goes through the entire corpus in this manner, evaluating $D_r$, specifying $\phi_1$, ... $\phi_n$ and selectively incrementing the $X_k$ and $Y_k$ registers.

### C. THE OPERATION OF THE FORMULA SYNTHESIZER

The input to the final machine program shown in Fig. 4, called *formula synthesizer,* is the set of tally counts in the $X_k$ and $Y_k$ registers. Its output is a valid maximal or nonmaximal basic algorithm, or a clear indication that important variables are missing from the list $\phi_1$, $\phi_2$, ... $\phi_n$.

The first operation performed by the formula synthesizer is the computation of a third set of numbers $\{Z_k\}$. For $X_k = 0$, $Z_k$ are undefined; for $X_k \neq 0$, $Z_k$ are defined as $Z_k = Y_k/X_k$. From the counting process, is follows that defined values of $Z_k$ satisfy $0 \leq Z_k \leq 1$. The $Z_k$ define the desired working formula $W_r$. It is convenient to discuss the synthesis of formulas in terms of four different types of patterns that can be described by the $Z_k$:

PATTERN TYPE 1*: All $Z_k$ are defined and either 0 or 1.*

When a pattern of this type is present, the formula synthesizer has found a maximal algorithm, one that cannot be improved insofar as the given text corpus is concerned. The vector of binary elements [$Z_1$, $Z_2$, $Z_3$, . . ., $Z_{2^n-1}$] is itself a representation of the desired working formula.* Since the $Z_k$ are all either *0* or *1,* each configuration corresponds to either doing or not doing the action, with no equivocation. The formula can be expressed in disjunctive canonical form by taking a sum of the logical products corresponding to the configurations for which $Z_k = 1$. Each product is obtained by conjoining all the *n* variables, negating just those to which a *0* is assigned in the configuration considered. For example, a simple hypothetical situation is illustrated in Table 2. The working formula corresponding to the $Z_k$ is $W_r = \sim \phi_1 \bullet \phi_2 \bullet \sim \phi_3 \vee \sim \phi_1 \bullet \phi_2 \bullet \phi_3 \vee \phi_1 \bullet \sim \phi_2 \bullet \phi_3$. Formulas thus obtained are in a so-called "canonical" disjunctive normal form. They can often be reduced to simpler normal forms by well-known rules of logic.[7,14,15]

Certain of the variables initially included in the list $\phi_1$, $\phi_2$, ... $\phi_n$ may not be needed in order to construct a valid working formula. Such variables will appear in the canonical form of a working formula only vacuously; they can be readily eliminated in the course of reducing the formula to a more minimal normal form.[17,18,19,20] For example, the formula $\sim \phi_1 \bullet \phi_2 \bullet \phi_3 \vee \sim \phi_1 \bullet \phi_2 \bullet \sim \phi_3$ contains the variable $\phi_3$ only vacuously and is reducible to $\sim \phi_1 \bullet \phi_2$. The logical rules for formula reduction are rigorous and machinable. A computer program that reduces formulas given in disjunctive canonical forms to more economical normal forms is being prepared at Harvard; it will contain provisions for eliminating vacuous variables.[21] There should be no difficulty connected with programming the necessary formula-reducing rules into the proposed formula-synthesizer.

* The methods for representing and reducing logical formulas mentioned in this section are well known in the fields of mathematical logic and algebraic switching theory. The basic logical principles are treated, for example, in Refs. 7, 14, 15, 16, and 17. Machinable methods for reducing logical formulas to minimal normal forms, for resolving "don't care" conditions, etc., are treated in Refs. 17, 18, 19 and 20.

TABLE 2

HYPOTHETICAL PATTERN OF $X_K$ AND $Y_K$ LEADING TO A PATTERN OF TYPE 1

| k | $\phi_1$ | $\phi_2$ | $\phi_3$ | $X_k$ | $Y_k$ | $Z_k$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 17 | 0 | 0 |
| 1 | 0 | 0 | 1 | 4 | 0 | 0 |
| 2 | 0 | 1 | 0 | 32 | 32 | 1 |
| 3 | 0 | 1 | 1 | 118 | 118 | 1 |
| 4 | 1 | 0 | 0 | 2 | 0 | 0 |
| 5 | 1 | 0 | 1 | 61 | 61 | 1 |
| 6 | 1 | 1 | 0 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 75 | 0 | 0 |

To the extent that the experimental corpus is only approximately representative of what can occur in Russian technical writing, so also will the algorithms synthesized from this data be only approximately valid. Before a machine-derived algorithm can be finally accepted, then, it must be subject to human scrutiny and tested further by a man-machine process like that discussed in Part 5 of this paper.

PATTERN TYPE *2: Defined $Z_k$ are either 0 or 1, but some $Z_k$ are undefined.*

A maximal algorithm can be synthesized when a pattern of this type is present, but it is not necessarily unique. The undefined $Z_k$ are in one sense like the so-called "don't care" conditions of switching theory.[17,18,19] Since configurations corresponding to these $Z_k$ do not occur in the experimental corpus, it might seem that *0*'s and *1*'s could be assigned to them in any desirable manner. In fact, machinable procedures exist for assigning values to $Z_k$ for "don't care" configurations in such a way as to simplify the resulting formula.[17,19,20] Assigning such values automatically in this somewhat offhand fashion would not, however, be a sound experimental procedure. Different formulas would result from assigning different sets of values to the undefined $Z_k$. While all such formulas would work equally well for the experimental corpus, they would behave differently in the event that one of the "don't care" conditions actually occurred in another text. If the value *1* were assigned to a $Z_k$, that should actually have the value *0,* then the algorithm would erroneously lead to the action $B_r$ whenever configuration *k'* is encountered in another text. To be safe, then, it is best to adopt a blanket rule for assigning values automatically; the machine is to assign the value *0* to each of the "don't care" $Z_k$. A synthesized algorithm will then *not* lead to the action $B_r$ if one of the "don't care" configurations is encountered in a later text.

Consideration is being given to the use of a ternary valued logic to enable better treatment of the "don't care" conditions. Assigning the value *0* to the undefined $Z_k$ is a "fail-safe" procedure, since the resulting algorithm leads to the execution of the action $B_r$ only in textual situations actually examined in the experimental corpus. Nevertheless, the effect of a *0* assigned to an undefined $Z_k$ is the same as that of a *0* computed from a nonvanishing $X_k$. Certain information is therefore not reflected in the algorithm: in the former case the configuration was not encountered, in the latter case it was encountered and found to have the value *0*. It may be possible to keep better track of this information by using a three-valued logic, where one of the values means "unresolved."

PATTERN TYPE 3: *Some of the $Z_k$ are proper fractions, $0 < Z_k < 1$, but at least one $Z_k$ is 1.*

A valid algorithm can be obtained when a pattern of this type is present, but the algorithm will be nonmaximal. The fractional values of $Z_k$ correspond to configurations that only sometimes lead to the given action in the experimental corpus. Other variables besides those included in $\phi_1, \ldots \phi_n$ must be taken into account when these configurations are present. The nonmaximal algorithm is obtained by simply rounding off each of the fractional $Z_k$ to zero, thus giving a pattern of type *1* or *2* that can be reduced by the methods already discussed.

Most of the algorithms that will be derived in the course of initial experiments with the formula finder will probably be nonmaximal. It is important to stress the fact that nonmaximal algorithms like maximal algorithms are "fail-safe" insofar as the experimental corpus is concerned. A derived algorithm leads to an action $B_r$ only for configurations that always lead to the action in the experimental corpus.

PATTERN TYPE *4: Some $Z_k$ are fractional and no $Z_k$ is 1.*

When a pattern of this type is present, no configuration of the given variables unambiguously leads to the given action and it is not possible to synthesize a valid basic algorithm from $\phi_1, \phi_2, \ldots \phi_n$ .

D. OUTPUTS OF THE FORMULA FINDER

The outputs of the formula finder are:

(1) The derived algorithm, in a readable format.
(2) The derived algorithm, in a machine-encoded format suitable as input to a trial translator system.
(3) An edited list of the configurations encountered, the corresponding $X_k$ and $Y_k$ counts, and the initial and final values of $Z_k$.

The first two outputs are only furnished when a pattern of type *1, 2,* or *3* is present; the third output is always produced. The function of the third output is to facilitate the human monitoring and control of the formula synthesizing process. The counts give an indication of the relative occurrence frequencies of the various configurations. They should enable linguists to evaluate an algorithm in terms of the types and frequencies of the situations encountered. When a pattern of type *2* is present, a linguist may find reasons to assign the value *1* instead of *0* to some of the undefined $Z_k$. The edited list of configurations should enable him to do so. When a pattern of type *3* is present, the edited list will clearly show the configurations with fractional $Z_k$. Inspection of the list might give insight into new variables that should be added to the list in order to obtain a maximal algorithm. When a pattern of type *4* is present, pertinent variables are clearly missing from the set $\phi_1, \phi_2, \ldots \phi_n$. Inspection of the edited list of $X_k$ and $Y_k$ might enable the identification of such variables.

A hypothetical list of configurations and $X_k$ and $Y_k$ values leading to the sample algorithm *(4)* is given in Table 3, where $\phi_1$ = N(i-l), $\phi_2$ = "K"(i—2), $\phi_3$ = PP(i-l), $\phi_4$ = PA(i-l), and $\phi_5$ = NF(i-l). The values of $X_k$ and $Y_k$ shown in the table are concocted to illustrate the formula reduction process; they

TABLE 3

HYPOTHETICAL TABLE OF $X_k$ AND $Y_k$ LEADING TO THE WORKING FORMULA $W_r = \phi_1 \bullet \sim \phi_2 \vee \phi_3 \vee \phi_4 \bullet \phi_5$ OF THE SAMPLE ALGORITHM ( SEE TEXT )

| k | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\phi_5$ | $X_k$ | $Y_k$ | Initial $Z_k$ | Change $Z_k$ | Final $Z_k$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 75 | 6 | .08 | Rnd | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 13 | 0 | 0 | | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 104 | 11 | .11 | Rnd | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 38 | 38 | 1 | | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 13 | 13 | 1 | | 1 |
| 5 | 0 | 0 | 1 | 0 | 1 | 17 | 17 | 1 | | 1 |
| 6 | 0 | 0 | 1 | 1 | 0 | 5 | 5 | 1 | | 1 |
| 7 | 0 | 0 | 1 | 1 | 1 | 11 | 11 | 1 | | 1 |
| 8 | 0 | 1 | 0 | 0 | 0 | 81 | 0 | 0 | | 0 |
| 9 | 0 | 1 | 0 | 0 | 1 | 37 | 4 | .11 | Rnd | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 63 | 19 | .30 | Rnd | 0 |
| 11 | 0 | 1 | 0 | 1 | 1 | 34 | 34 | 1 | | 1 |
| 12 | 0 | 1 | 1 | 0 | 0 | 3 | 3 | 1 | | 1 |
| 13 | 0 | 1 | 1 | 0 | 1 | 28 | 28 | 1 | | 1 |
| 14 | 0 | 1 | 1 | 1 | 0 | 35 | 35 | 1 | | 1 |
| 15 | 0 | 1 | 1 | 1 | 1 | 60 | 60 | 1 | | 1 |
| 16 | 1 | 0 | 0 | 0 | 0 | 186 | 186 | 1 | | 1 |
| 17 | 1 | 0 | 0 | 0 | 1 | 107 | 107 | 1 | | 1 |
| 18 | 1 | 0 | 0 | 1 | 0 | 91 | 91 | 1 | | 1 |
| 19 | 1 | 0 | 0 | 1 | 1 | 62 | 62 | 1 | | 1 |
| 20 | 1 | 0 | 1 | 0 | 0 | 43 | 43 | 1 | | 1 |
| 21 | 1 | 0 | 1 | 0 | 1 | 111 | 111 | 1 | | 1 |
| 22 | 1 | 0 | 1 | 1 | 0 | 136 | 136 | 1 | | 1 |
| 23 | 1 | 0 | 1 | 1 | 1 | 72 | 72 | 1 | | 1 |
| 24 | 1 | 1 | 0 | 0 | 0 | 194 | 107 | .55 | Rnd | 0 |
| 25 | 1 | 1 | 0 | 0 | 1 | 109 | 72 | .66 | Rnd | 0 |
| 26 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | - | Set | 0 |
| 27 | 1 | 1 | 0 | 1 | 1 | 26 | 26 | 1 | | 1 |
| 28 | 1 | 1 | 1 | 0 | 0 | 13 | 13 | 1 | | 1 |
| 29 | 1 | 1 | 1 | 0 | 1 | 81 | 81 | 1 | | 1 |
| 30 | 1 | 1 | 1 | 1 | 0 | 30 | 30 | 1 | | 1 |
| 31 | 1 | 1 | 1 | 1 | 1 | 19 | 19 | 1 | | 1 |

probably bear little resemblance to those that would actually be found in texts. The initial set of $Z_k$ forms a pattern of type *3*. The final column shows the results of rounding fractional values of $Z_k$ to *0,* and assigning the value *0* to the undefined $Z_{26}$. The canonical normal form of the resulting $W_r$ formula is too long to be listed here; it involves a sum of twenty-three terms, each being a product of the five variables. When reduced to a minimal normal form it becomes $\phi_1 \bullet \sim \phi_2 \vee \phi_3 \vee \phi_4 \bullet \phi_5$, the working formula of the desired nonmaximal algorithm,

## 5. The Feedback System for Research in Automatic Language Translation

The formula finder is one of the three components of a proposed man-machine feedback system for research in automatic language translation. The other two components are the trial translator[2] and the monitoring human linguists. The over-all feedback system is block-diagrammed in Fig. 6. Three main feedback loops are shown in the diagram; they are labeled $L_1$, $L_2$, and $L_3$. The derivation of an algorithm starts with loop $L_1$. The humans initially suggest clues to the formula finder: $D_r$, $B_r$, and $\phi_1$, $\phi_2$, ... $\phi_n$. The outputs of the formula finder are examined by the linguists. If no basic algorithm is found or if the machine-derived algorithm is unacceptable, the set of variables may be modified and the formula finding run repeated. This iterative process, corresponding to loop $L_1$, can be repeated until an algorithm is tentatively accepted for further testing.

Once an automatically synthesized algorithm is tentatively accepted, the iterative process of loop $L_2$ is called into play. The machine-coded version of the derived algorithm is used by the trial translator to produce experimental improved translations of Russian texts. The linguists examine these translations, and perhaps suggest further improvements or changes in the algorithm. The improved algorithm is then fed back into the trial translator, new experimental translations are produced and examined, etc.

A second type of feedback may also be employed within loop $L_2$. Linguists may know of involved grammatical situations not represented in the experimental corpus, but pertinent to algorithms being studied. They may therefore wish to devise special sentences containing problematic constructions, or to draw such sentences from reference grammars. The task of automatic formula finding would be greatly complicated if such sentences were simply added to the experimental corpus. Nevertheless, for the purpose of subjecting algorithms known to be basically sound to extreme conditions, the linguists may wish to use the proble-

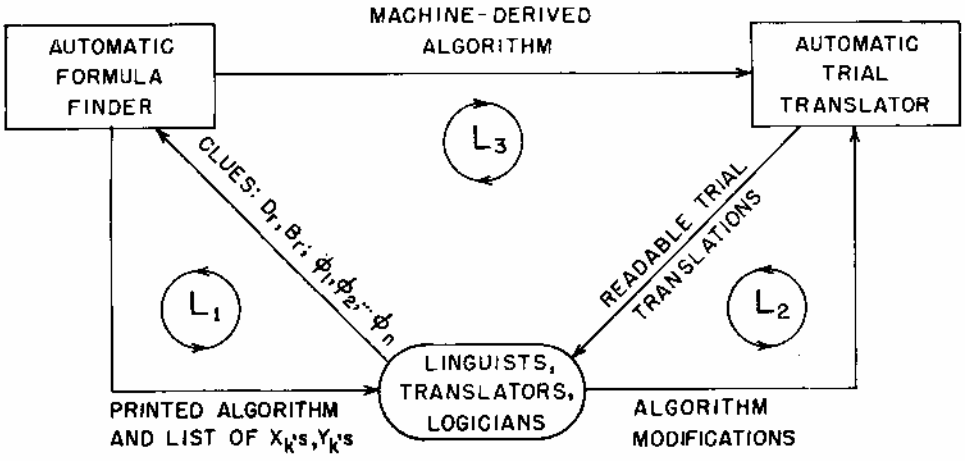



FIGURE 6
A MAN-MACHINE FEEDBACK SYSTEM FOR RESEARCH IN AUTOMATIC LANGUAGE TRANSLATION

matic sentences in the final testing stages of loop $L_2$ Hopefully, the iterative process of loop $L_2$ will thus provide for "vernier" adjustments leading finally to a valid and useful algorithm.

Feedback loop $L_3$, might play a role in going from one basic algorithm to another. The trial translator can produce translations reflecting the product transformation $T_t = A_1A_2, \ldots, A_q$ of any number of known and tested algorithms. It can be safely assumed that for a long time $T_t$ will fall far short of the complete syntactic and semantic transformation $T_s$ performed by the human post-editor. In the course of time, more and more algorithms $A_{q+1}, A_{q+2}$, etc., will be added to $T_t$. At any given time, the printed translation resulting from $T_t$ will contain only residual ambiguities that should stand out in bold relief. By inspecting the partially improved translations resulting from a given $T_t = A_1A_2, \ldots, A_q$, then, linguists might be able to divine clues about a basic algorithm $A_{q+1}$ that should naturally be derived next. It may be possible to express these clues in the form of $D_r$, $B_r$, and $\phi_1, \phi_2, \ldots \phi_n$ statements. If so, the clues may be fed into the formula finder, and another algorithm found through the processes of loops $L_1$ and $L_2$.

The machine programs of the proposed formula finder must still be written, and some of the manual procedures must be worked out in greater detail; many interesting questions about automatic formula finding still remain essentially unsolved. At present, algorithms are being successfully found by more traditional methods of scholarly insight, with the machine playing a more subordinate role than that illustrated in Fig. 6. Nevertheless, the writer feels that automatic formula finding is potentially a fruitful area for further research in automatic language translation. The logical techniques suggested in this paper can readily be adopted for formula finding with other pairs of natural or artificial languages. The writer also believes that these techniques can be extended and used for research in several allied fields, such as automatic speech and pattern recognition, and the empirical study of sequential automata.

*Received August, 1959*

## References


1. Yngve, V. H., "A Programming Language for Mechanical Translation," *Mechanical Translation,* Vol. 5, No. 1, pp. 25-41 (1958).
2. Giuliano, V. E., "The Trial Translator, an Automatic Programming System for the Experimental Machine Translation of Russian to English," to appear in the *Proceedings of the 1958 Eastern Joint Computer Conference.*
3. Oettinger, A. G., Foust, W., Giuliano, V. E., Magassy, K., and Matejka, L., "Linguistic and Machine Methods for Compiling and Updating the Harvard Automatic Dictionary," *Preprints of Papers for the International Conference on Scientific Information,* National Academy of Sciences, National Research Council, Washington, D. C., Part V, pp. 137-160 (1958).
4. Giuliano, V. E., *An Experimental Study of Automatic Language Translation,* Doctoral Thesis, Harvard University (1959).
5. Giuliano, V. E., and Oettinger, A. G., "Research in Automatic Translation at the Harvard Computation Laboratory," to appear in the proceedings of the International Conference on Information Processing held in Paris in June, 1959, under UNESCO sponsorship.
6. Matejka, L., "Grammatical Specifications in the Russian-English Automatic Dictionary," *Design and Operation of Digital Calculating Machinery,* Report No. AF-50, Harvard Computation Laboratory, Section V (1958).
7. Quine, W. V., *Methods of Logic,* Henry Holt and Co., New York (1953).
8. Fargo, N., and Rubin, J., "Tentative Statement of Reasons for Choice in the Translation of Noun Suffixes," *Seminar Work Paper: MT-40* (mimeographed), Georgetown University Institute of Languages and Linguistics (1957).
9. Mattingly, I., Manuscript to appear in *Papers Presented at the Seminar on Mathematical Linguistics,* Vol. V, (1959).
10. Oettinger, A. G., A *Study for the Design of an Automatic Dictionary,* Doctoral Thesis, Harvard University (1954).
11. Giuliano, V. E., "Programming an Automatic Dictionary," *Design and Operation of Digital Calculating Machinery,* Report No. AF-49, Harvard Computation Laboratory, Section I (1957).
12. Berson, J. S., "The Calculus of Propositions in Automatic Translation," *Papers Presented at the Seminar on Mathematical Linguistics,* Vol. IV, on deposit at Widener Library, Harvard University (1958).
13. Burks, A. W., Warren, D. W., and Wright, J. B., "An Analysis of a Logical Machine Using Parenthesis-free Notation," *Mathematical Tables and Other Aids to Computation,* Vol. 8, Nos. 45-48, pp. 53-57 (1954).
14. Quine, W. V., *Mathematical Logic,* Harvard University Press, Cambridge (1947).
15. Copi, I., *Symbolic Logic,* MacMillan, New York (1954).
16. Keister, W., Ritchie, A. E., and Washburn, S., *The Design of Switching Circuits,* Van Nostrand Co., New York (1951).
17. *Synthesis of Electronic Computing and Control Circuits,* Annals of the Computation Laboratory of Harvard University, Vol. 27, Harvard University Press, Cambridge (1951).
18. Quine, W. V., "Two Theorems About Truth Functions," *Boletin de la Sociedad Matematica Mexicana,* Vol. 10, pp. 64-70 (1953).
19. Karnaugh, M., "Synthesis of Combinational Logic Circuits," *Communication and Electronics,* No. 9, pp. 593-598 (November 1953).
20. Roth, J. P., "Algebraic Topological Methods in Synthesis," *Proceedings of an International Symposium on the Theory of Switching,* Annals of the Computation Laboratory of Harvard University, Vol. 29, Harvard University Press, Cambridge (1959).
21. Roche, J. W., Solomon, N. B., and Kelley, K. A., manuscript to appear in *Theory of Switching,* Report No. BL-23, Harvard Computation Laboratory (1959).