MECHANICAL TRANSLATION

DEVOTED TO THE TRANSLATION OF LANGUAGES WITH THE AID OF MACHINES

VOLUME FIVE, NUMBER ONE JULY, NINETEEN FIFTY EIGHT

News

UNIVERSITY OF CALIFORNIA, BERKELEY

A group has started work on the mechanical translation of Russian at the University of California, Berkeley. The work will be under the direction of M. Haas, L. Henyey and S. Lamb, and it will be a cooperative effort of the Computer Center and the Department of Linguistics. The group is planning to analyze a very large corpus of Russian text and to formulate a translation program on the basis of the analysis. Machine methods will be used extensively in the analysis.

WAYNE STATE UNIVERSITY

H. Josselson and A. Jacobson have formed a mechanical translation group at Wayne State University. They will be concerned primarily with the translation of Russian into English in the field of astrophysics. The direction which the program intends to pursue differs from other mechanical translation projects in its emphasis on statistical observations. Thus, the detailed rules for translation will be worked out on a conditional probability basis; pertinent data will be stored in a computer in a manner most economical from the point of view of access time; the discrete linguistic units pertinent to translation will be classified and arranged with the help of statistical observations; and mathematical models of language will be constructed to insure both validity and predictability for future performance.

THE UNIVERSITY OF TEXAS

Interest in mechanical translation has been increasing at the University of Texas. This semester W. Lehmann, S. Werbow and W. Winter are giving a seminar on the topic of the mechanical translation of German. Besides the six students who are registered for the seminar, there are six regular visitors and some occasional visitors who attend the meetings. Members of the seminar are being encouraged to prepare and present preliminary work papers.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

The M.I.T. group is planning a second Summer Workshop on German to English mechanical translation. Its purpose is to provide an informal and stimulating atmosphere in which experienced people as well as those who are new to the field can exchange ideas and work on various problems in German and English morphology and syntax from the mechanical translation point of view.

The M.I.T. group is also planning for the spring or early summer a short course in programming using the COMIT system.

Those who are interested in attending either of these programs should write to the undersigned.

Victor H. Yngve
Room 20B-101D
M. I. T.
Cambridge, Mass.

An Input Device for the Harvard Automatic Dictionary†

Anthony G. Oettinger, Computation Laboratory,
Harvard University, Cambridge, Massachusetts

A standard input device has been adapted to permit transcription of either Roman or Cyrillic characters, or a mixture of both, directly onto magnetic tape. The modified unit produces hard copy suitable for proofreading, and records information in a coding system well adapted to processing by a central computer. The coding system and the necessary physical modifications are both described. The design criteria used apply to any automatic information-processing system, although specific details are given with reference to the Univac I. The modified device is performing satisfactorily in the compilation and experimental operation of the Harvard Automatic Dictionary.

THE PROPERTIES of a given automatic information-processing machine depend primarily on the algorithms the machine is capable of applying to the tokens [1] for the abstract elements it is said to process. Configurations of the states of sets of two-state devices, or pulse trains where pulses are present or absent in definite time intervals, are commonly used as tokens in contemporary machines. Abstract elements, e.g., the integers, are named by symbols of various kinds. For example, the numerals "2", "II", and "10" all name the number 2. Likewise, various symbols can be used to name tokens. It is a useful and widely accepted convention to use the symbol "0" as the name for one state of a two-state device, and the symbol "1" as a name for its other state. Frequently, the symbols "0" and "1" are used also as binary numerals. In a context where both these usages occur, a string such as "1001" functions homographically both as a name for the number 9 and as a name for a particular configuration of a set of four two-state devices. This practice is confusing in discourse about machines intended for or adapted to purposes other than numerical computation, especially when the relation between machine tokens and abstract elements is the chief subject of discussion. In this paper, therefore, "0" and "1" will be used exclusively as the names of tokens.

The mapping between machine tokens and the abstract elements a given machine is said to process can be regarded as defined by the input and output hardware of the machine. For example, if a pulse train 1010100 is to be regarded as a token for the letter A, it is desirable to arrange matters so that such a pulse train will cause a printer to print the literal "A". When an order relation exists among the tokens in a machine, as imposed, for example, by comparison and branch instructions, and when the abstract elements themselves are an ordered set, it is usually desirable to relate abstract elements and tokens by an order-preserving mapping. For example, in a machine designed to recognize 1010100 to be "smaller" than 0010101 and 0010101 in turn to be smaller than 0010110, the mapping A — 1010100, B — 0010101, C — 0010110 preserves normal alphabetic order, whereas A — 0010101, B — 1010100, C — 0010110 does not.

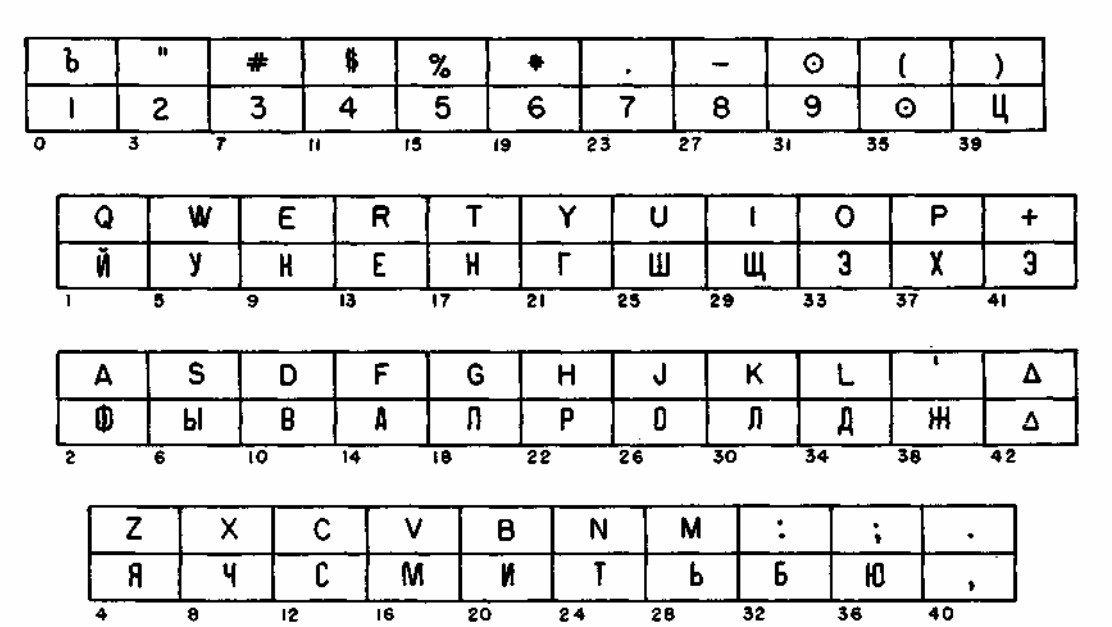
---

† This work has been supported in part by the Harvard Foundation for Advanced Study and Research, the United States Air Force, and the National Science Foundation.

1. This term was originated by C. S. Peirce. For an explanation of the underlying distinctions, see H. Reichenbach, Elements of Symbolic Logic, Macmillan, New York, 1947, p.4.

The Univac I computer is currently in use at the Harvard Computation Laboratory in connection with the development of an operating automatic dictionary[2] and for basic research on the problems of automatic translation from Russian into English. The normal mapping between numbers, letters of the Roman alphabet, punctuation marks, and other standard symbols on the one hand, and machine tokens on the other, is given in Figure 2 by the columns headed "Upper Case" and "Binary Code" (except for key no. 0). This mapping is established by all input and output devices associated with the machine, in particular by the Unityper, which is used to record information onto magnetic tape, and by the High-Speed Printer, which is the major output unit. Thus, when an A is typed, a token 1010100 is recorded, and such a token will in turn cause the High-Speed Printer to print an A.

Adapting a machine like the Univac to handle Cyrillic letters is conceptually a trivial matter. To permit alphabetization of Cyrillic material, an order-preserving mapping between the Cyrillic alphabet and Univac tokens is necessary. Many such mappings can readily be established. Once this has been done, the internal operation of the machine with Cyrillic material presents no difficulties. However, unless the input and output devices are physically altered, certain practical problems obviously arise.



Keyboard Layout

Figure 1

As a first step, it is simple to cover the keys on the Unityper with keytops labelled with Cyrillic letters. From the point of view of typing ease and accuracy the most desirable keyboard layout (Fig. 1) is one in standard use on ordinary Cyrillic typewriters. Unfortunately, merely replacing keytops solves only a part of the practical problem. First, the typewriter

2. Oettinger, A. G., Foust, W., Giuliano, V., Magassy, K., Matejka, L., "Linguistic and Machine Methods for Compiling and Updating the Harvard Automatic Dictionary" (To be presented at the International Conference on Scientific Information, Washington D.C., November 1958, and published in the Proceedings of the conference).

KEY #	LOWER CASE	UPPER CASE	BINARY CODING 1	2	3	4	5	6	7
0	Ю	Б	0	1	0	1	0	0	1
1	Й	;	0	1	0	1	0	1	1
2	Ф	М	1	1	0	1	0	1	0
3		2	1	1	0	0	0	0	1
4	Я	У	0	1	1	1	1	0	0
5	У	К	0	1	0	1	0	0	1
6	Ш	Т	1	1	1	0	1	0	1
7		3	1	0	0	1	1	1	0
8	Ч	З	1	1	0	1	0	1	0
9	Н	В	0	0	1	1	0	0	0
10	В	3	1	0	0	0	1	1	0
11		4	0	1	0	0	1	1	1
12	С	)	0	0	1	0	0	1	1
13	Е	6	1	0	1	0	0	0	1
14	А	1	0	0	0	1	1	0	0
15		5	0	0	1	1	0	0	0
16	М	Д	1	0	1	1	0	1	0
17	Н	Е	1	1	0	1	0	0	0
18	П	Н	1	0	0	1	1	1	0
19		6	1	1	0	0	0	1	0
20	И	9	1	0	0	0	1	0	0
21	Г	4	0	0	0	0	1	1	1
22	Р	1	0	0	1	1	0	0	1
23		7	1	0	0	1	0	1	0
24	Т	К	0	1	0	0	1	0	0
25	Щ	Р	0	0	1	0	1	0	1
26	О	Г	0	1	0	0	1	0	0
27		8	0	0	0	0	0	1	0
28	Б	У	0	1	1	0	0	1	1
29	Ц	+	1	0	1	0	1	0	0
30	Л	С	0	0	1	0	0	1	1
31		9	1	1	0	0	0	1	0
32	Ь	2	1	0	0	1	0	1	0
33	З	8	0	0	0	1	1	0	1
34	Д	5	0	0	0	1	0	0	0
35		0	1	0	0	0	0	1	1
36	Ю	Х	1	0	1	1	0	0	1
37	Х	Н	1	0	0	1	0	0	0
38	Ж	7	1	0	0	0	1	1	0
39	Ц	Р	0	0	1	0	0	1	1
40		,	1	1	0	1	0	0	0
41	З	В	0	1	1	1	0	0	1
42		Δ	0	0	0	0	0	0	0
43	TRIP		0	0	0	0	0	0	0

NOTES:

1) 0 = NOTCH CUT, 1 = NO CUT

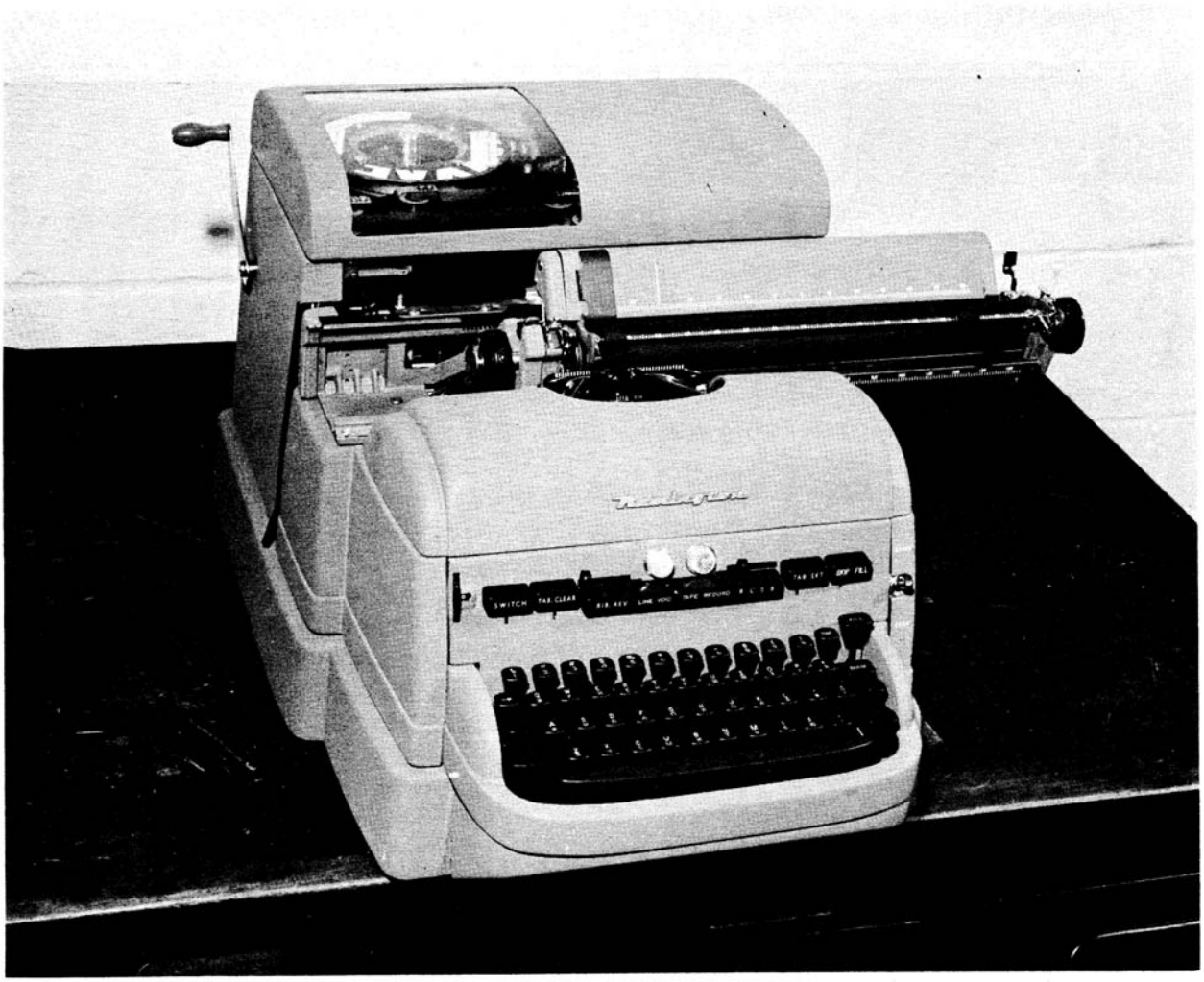
2) WHEN THE CHARACTER PRINTED ON THE HARD COPY IS DIFFERENT FROM THAT RECORDED ON MAGNETIC TAPE, THE FORMER IS SHOWN TO THE SIDE OF THE APPROPRIATE CIRCLE.

Definition of Mappings

Figure 2

continues to print Roman letters (e.g., Q for Й), a cryptographic transformation that makes proofreading most difficult. Second, the correspondence between the Cyrillic alphabet and machine tokens established in this way does not preserve Cyrillic alphabetic order. To reconcile these conflicting demands, a composition of two successive mappings can be used. [3] The first, established by the input device with covered keytops, leads to the representation of Cyrillic information in a "typewriter code." A subsequent code conversion is made automatically on the computer, at the expense of some running time, leading to the representation of Cyrillic letters in a "ranked code." The resultant mapping is order-preserving. In Figure 2, the Cyrillic letters are named in the "Lower Case" column. The token corresponding to a particular Cyrillic letter in the ranked code is named in the "Binary Coding" column, in the same row as the letter. The choice of this particular mapping was made for technical reasons

3. Ibid.



Modified Roman / Cyrillic Unityper

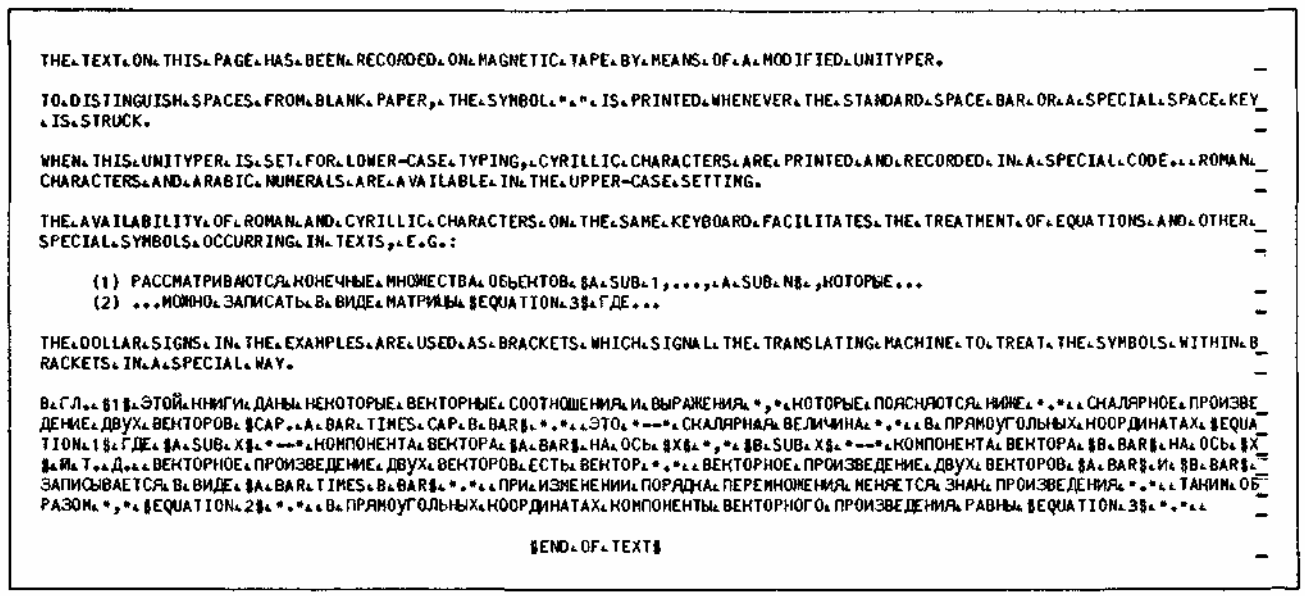
Figure 3

described in detail elsewhere.[4] Similar expedients have been used by others.[5]

Recently, we modified a standard Unityper to enable both the direct conversion from Cyrillic to ranked code, and the production of Cyrillic hard copy. The necessity for a costly intermediate code conversion by the computer itself is thereby eliminated, and proofreading is made relatively easy. The layout of the keyboard of the modified typewriter is shown in Figure 1. Figure 3 is a photograph of the actual machine. A sample of the hard copy produced by the modified Unityper is shown in Figure 4. The facility for interspersing standard and Cyrillic symbols is proving extremely useful in the recording of Russian texts, as illustrated in Figure 4.

4. Giuliano, V., "Programming an Automatic Dictionary" Design and Operation of Digital Calculating Machinery, Progress Report AF-49, Harvard Computation Laboratory, 1957, pp. I-42-I-45.

5. Edmundson, H.P., Hays, D.G., Renner, E.K., Button, R.I., "Manual for Keypunching Russian Scientific Text" RM-2061, RAND Corporation, 1957.



Demonstration Hard Copy Produced by the Modified Unityper

Figure 4

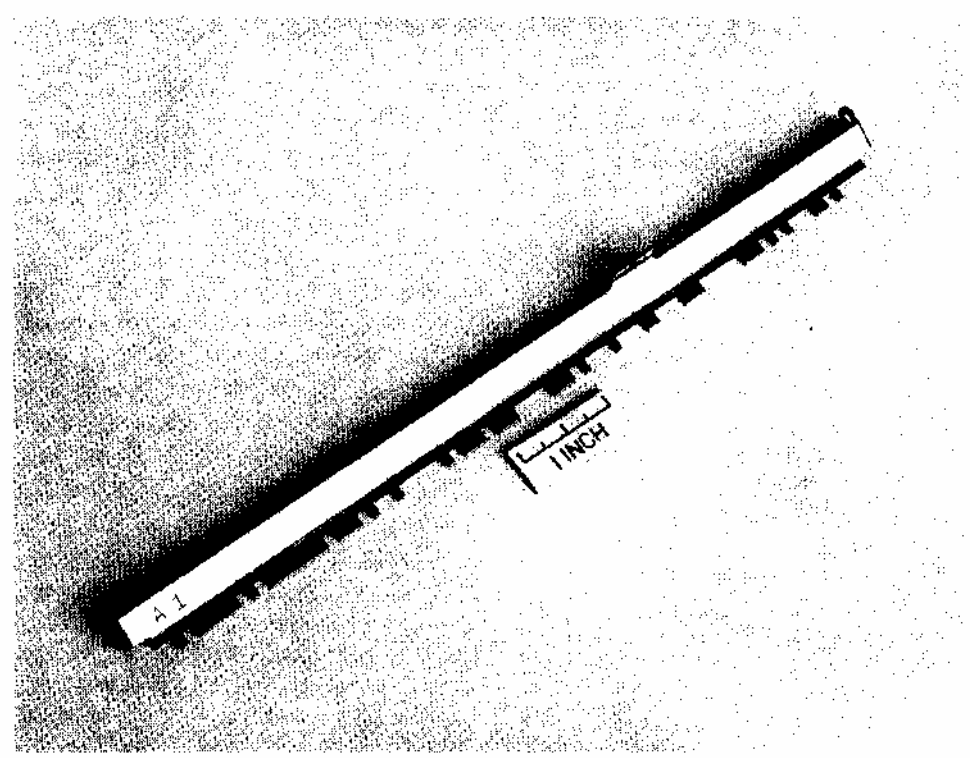
In lower case, the typewriter is Cyrillic. Except for three of the very low frequency letters, the layout is standard. In upper case, the typewriter functions as a standard model, except for the absence of a few special symbols normally available, and for the presence of one infrequently used Cyrillic letter. The mapping which obtains when the typewriter is in upper case is described by the "Upper Case" and "Binary Coding" columns of Figure 2. For example, 1101011 is a token for the letter Q. In lower case, the mapping is that described by the "Lower Case" and "Binary Coding" columns. For example, 0010011 is defined as a token for the Cyrillic letter Й.

The symbols circled in the "Lower Case" column are the normal correspondents of the tokens. For example, while 0010011 is defined as a token for Й in the ranked code, it is normally a token for the semi-colon. Therefore, since the output equipment has not been modified, Cyrillic material in the ranked code still would print in cryptographic form, e.g., "56EU" for "ДЕНЬ" A fast transliteration routine developed by Andrew Kahr for converting ranked code into a standard transliteration code has proved satisfactory for experimental purposes. It yields, for example, "DEN'" for "ДЕНЬ" .

Relatively few physical changes were necessary to achieve the desired modifications. Specially prepared keytops labelled as in Figure 2 had to be substituted for the normal ones. Corresponding type slugs were not available on the market, but were cast by the manufacturer from dies specially cut to our specifications. The correspondence between typewriter keys and the machine tokens is established physically by a set of encoding bails, notched in the pattern described in Figure 2. A photograph of the bail associated with the leftmost column of binary coding (Column 1) is shown in Figure 5. These bails were cut in our shop from blanks provided by the manufacturer, who undertook to harden the cut bails to his own specifications. Instal-

ling keytops, type slugs, and bails presented no unusual difficulties.

The author wishes to express his appreciation to the Remington Rand Univac Division of Sperry Rand Corporation, in the persons of Messrs. Edward L. Fitzgerald and Ted Carp, for their cooperation, especially in casting type slugs to our specifications, and to Messrs. Allen Christensen and Daniel Spillane of the Staff of the Computation Laboratory for machining the bails.



An Encoding Bail

Figure 5

Research Methodology for Machine Translation

H. P. Edmundson and D. G. Hays, The RAND Corporation, Santa Monica, California

The general approach used at The RAND Corporation is that of convergence by successive refinements. The philosophy that underlies this approach is empirical. Statistical data are collected from careful translation of actual Russian text, analyzed, and used to improve the program. Text preparation, glossary development, translation, and analysis are described.

Introduction

THIS PAPER is the first of a series that describes the methods now in use at The RAND Corporation for research on machine translation (MT) of scientific Russian. The limitation to scientific text results from the importance of prompt, widespread distribution of Soviet scientific literature in the United States. The purpose of this series is to clarify the technical problems of computer application in linguistic research, to stimulate research in machine translation, and to encourage standardization of working materials. The present paper describes the general approach being followed, giving its philosophy and method.

The general approach used at The RAND Corporation for conducting research on MT is that of convergence by successive refinements. At each stage, automatic computing machinery is used for some aspects of translation, and for collecting and analyzing data about other aspects,

The philosophy that underlies this approach is empirical, in the sense that statistical data are collected from careful translations of actual Russian text, analyzed, and used to improve the MT program. Preconceptions about language are generally suppressed in this approach; no attempt is made to create a complete linguistic theory in advance. Nevertheless, cogent formalizations and previous knowledge of language are adopted whenever they seem useful.

The method is conveniently divided into four components:

1. Text Preparation. Russian scientific articles are pre-edited and punched into a deck of IBM cards.

2. Glossary Development. A second deck is punched, including a card for every different "word" in the text. Some pertinent linguistic information is added.

3. Translation. Using the glossary, an IBM 704 program produces a rough translation of the text. This translation is postedited.

4. Analysis. The postedited translation is studied in order to improve the glossary and the machine-translation program.

These four components of the research method are described in some detail in the present paper (see pp. 10 to 15 and Fig. 1). However, a complete exposition is contained in the RAND Studies in Machine Translation, nos. 3 through 9.

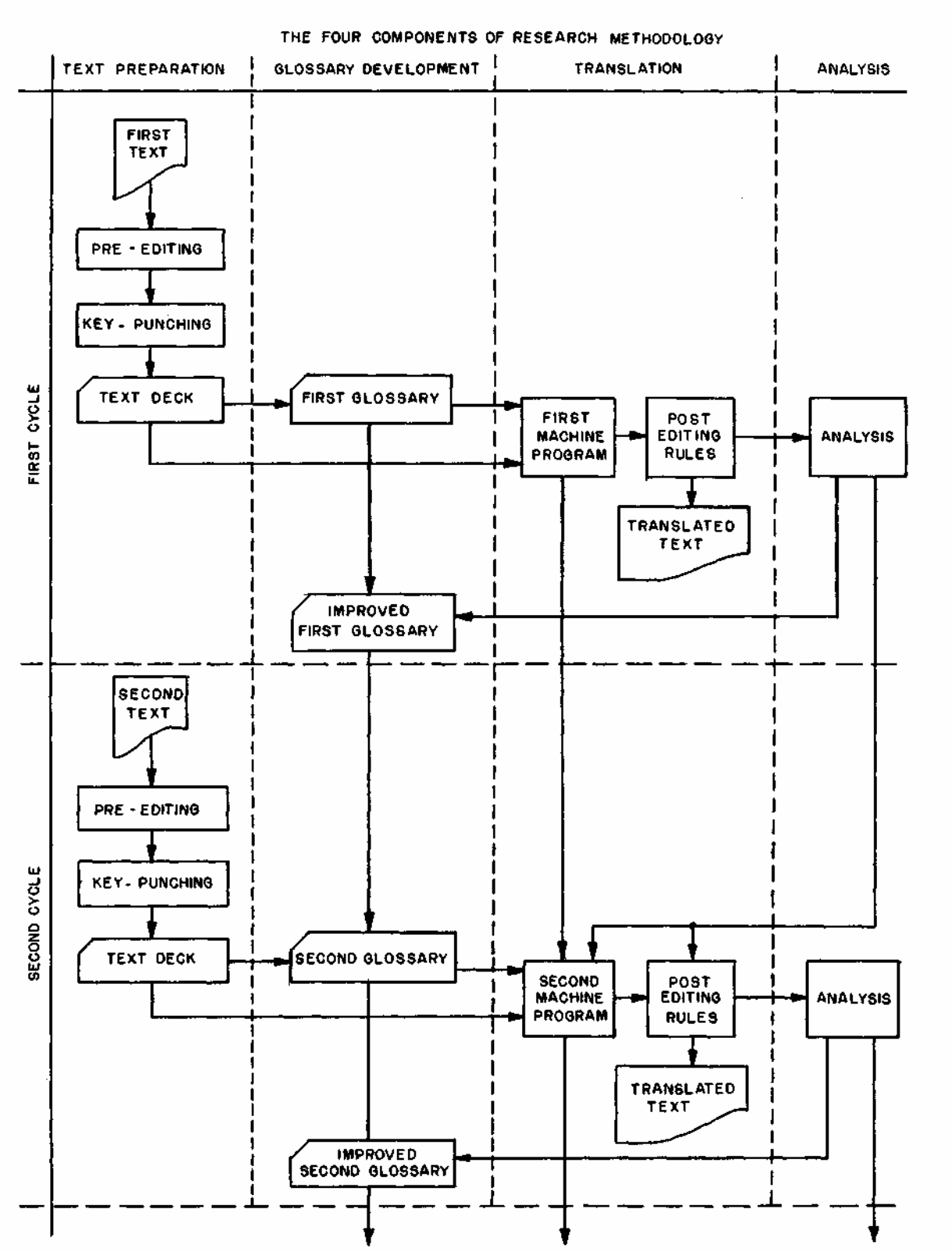
Some Definitions

It is necessary to be clear concerning the meanings of certain words that we shall use in a technical sense. This research employs a number of distinctions that are common only among linguists, and that accordingly call for special definitions.

Corpus: a group of articles or books selected for analysis.

Form: a distinctive sequence of characters. Thus every change in spelling is a change in form; "photon" and "photons" are different forms of the same word.

Occurrence (of a form): a sequence of printed characters, in a corpus, preceded and followed by either spaces or punctuation. An occurrence is identified by its ordinal position in the corpus. Hence, by definition, "photon" on page 1 and "photon" on page 2 are different occurrences of the same form.



FLOW CHART OF THE RESEARCH PROCESS

FIGURE 1

Word: a form that represents a set of forms differing only in inflection. For example, "great" and "greater" are forms of the same word, while "great* and "large" are forms of different words.

Glossary (of a corpus): a list of all the forms that occur in a corpus; grammatical and semantic information may also appear.

Dictionary (of a language): a list of all the words in the language, each represented by one form; grammatical and semantic information may also appear. A dictionary changes as the language expands and contracts.

These distinctions are necessary for precise study of language; they are used, as consistently as possible, throughout this work. Additional terms are introduced as required.

Text Preparation

The preparation of a corpus of Russian scientific text on punched cards involves selection of articles, pre-editing, design of machine codes and card formats, and keypunching.

1. Selection of Articles

The present RAND corpus consists of articles in the fields of physics and mathematics. These fields were chosen because of their importance for national security, and also because of the fact that their reputedly limited vocabularies assure a slow rate of glossary increase, which is useful in the preliminary cycles of research. Two journals are represented: Sections of the Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki, which had been keypunched in a research project at the University of Michigan, furnish a valuable beginning;* in addition, articles from the Doklady Akademii Nauk SSSR are being keypunched at RAND, so that the two journals can be compared for vocabulary and sentence structure. Within the Doklady, selection is made by a scientist on the basis of substantive interest and high ratio of text to symbols and equations. A bibliography of the current RAND corpus is contained in MT Study 9.[1]

2. Pre-editing

Pre-editing is necessary for efficient keypunching; decisions are made before the keypunch operation begins, so that the operator knows exactly what to punch and in what order. The variety of characters and arrangements that is possible on a printed page cannot be reproduced on a standard keypunch machine. The pre-editor substitutes, for each nonpunchable symbol or formula, a code that can be punched. He assigns and index number to each article; to each page of the article; to each line of the page; and to each occurrence in the line. The current rules for pre-editing are contained in MT Study 4. [2]

3. Machine Codes

American punched-card machinery is not designed to process the Cyrillic alphabet; modifications are required, either in equipment or in procedure. For the present, it is most convenient to adapt procedures. Accordingly, three distinct codes for the Cyrillic alphabet are needed:

a) Keypunch Code. Special key-tops are prepared for the Cyrillic alphabet, and arranged on the keyboard of an IBM Type 026 keypunch in the pattern of a standard Russian typewriter. Each letter of the Cyrillic alphabet is punched into cards with a unique combination of holes, but these combinations are not adapted to machine sorting or listing.

b) Sort Code. The standard construction of IBM card sorting and collating machines defines a natural ordering of certain punch combinations. The RAND sort code assigns these punch combinations to the Cyrillic characters in their natural order. Thus it is possible, using standard IBM machines and standard procedures, to sort cards into Cyrillic alphabetic order.

c) List Code. The letters of the Roman alphabet, decimal digits, and a few special characters can be printed on IBM equipment. Each of these characters is printed by a unique punch combination. The RAND list code causes IBM equipment to print a Roman transliteration of the Cyrillic original. The transliteration used here was designed for convenient machine printing.

---

* Andreas Koutsoudas, the director of the Michigan project, has contributed to this RAND study as a consultant.

1. H.P. Edmundson, K.E. Harper, D.G. Hays, and A. Koutsoudas, "Studies in Machine Translation—9: Bibliography of Russian Scientific Corpus," in preparation.

2. H.P. Edmundson, D.G. Hays, E.K.Renner, and R.I.Sutton, "Studies in Machine Translation — 4: Manual for Pre-editing Russian Scientific Text," in preparation.

Of these three codes, the sort code seems most reasonable as a permanent, standard IBM code for Cyrillic characters. In the first place, the "natural" order of the punch combinations is related to the arrangement of punches in the card column, as well as to the construction of sorters and collators. Furthermore, the sort code uses one column for each Cyrillic character, whereas the list code requires as many as four columns for phonetic representations of some characters.

The keypunch code can be eliminated by mechanical alteration of the keypunch. The list code can be eliminated by construction of typewheels with Cyrillic characters for the machines used in listing. In the absence of special equipment, use of three distinct codes is unavoidable; conversions among the codes are most conveniently performed on an automatic computer.

4. Card Formats

Each occurrence of a form in the corpus, as marked by the pre-editor, is punched into an IBM card. This card contains a sequence number indicating the order of the occurrence in the corpus, punctuation marks before and after the occurrence, and the Russian form of the occurrence.

In order to record all of the information needed in translation and analysis, two cards are required for each occurrence. Both cards contain the information listed above. In addition, the first card (the translation text card) contains glossary information (see Glossary Development); the second card (the analytic text card) contains analytic information (see Translation and Analysis).

Complete descriptions of machine codes and card formats are contained in MT Study 3.[3]

Glossary Development

In accordance with the general approach of this project, the glossary is developed by increments. An initial glossary is prepared from a small corpus; examination of a new corpus leads to expansion of this glossary; and so on. Initially, the rate of growth of the glossary is large; as the process continues, the rate will decrease, but never vanish.

During each cycle, the new corpus is alphabetized on the Russian form. A summary deck is produced, containing one card for each different form; the number of occurrences of each form is recorded in this process. The new sum mary deck is mechanically matched with the old glossary, and new forms are listed for coding by linguists.

The linguist adds information to the new glossary cards as follows:

a) Grammar Code. Each form is coded for part of speech, case, number, gender, tense, person, degree, and so forth. The current RAND code has more than 1000 categories; it is described in MT Study 6.[4]

b) Word Number. Each form in the corpus is numbered automatically; it remains for the linguist to collect all inflected forms of a single word and assign a number identifying the group as a word. (See MT Study 7.)[5]

c) English Equivalents. If the new form is a form of a word in the old glossary, the English equivalents previously used are carried forward. If no form of the word has occurred before, the linguist assigns up to 3 tentative English equivalents. (See MT Study 7.)[5] His selection may be altered after postediting. (See Analysis.)

Grammar code, word number, and English equivalents are keypunched into the summary cards and then transferred to the translation text cards.

Translation

From one point of view, almost the whole research process consists of translation. In a stricter sense, however, "translation" is used to describe the two-stage process of machine translation and postediting. The process begins with the translation text deck, already containing glossary information and sorted into textual order. A 704 program produces a listing of the text as a rough translation; a postedi-tor works on this list, converting it into a smooth English version of the Russian original.

---

3. H.P.Edmundson, D.G.Hays, and R.I.Sutton, "Studies in Machine Translation—3: Resume of Machine Codes and Card Formats," August 18, 1958.

4. K. E. Harper, and D. G. Hays, "Studies in Machine Translation—6: Manual for Coding Russian Inflectional Grammar, " March 3, 1958.

5. H.P.Edmundson, K.E.Harper, D.G.Hays, "Studies in Machine Translation—7: Manual for Assigning Word Numbers and English Equivalents to Russian Forms," in preparation.

The object of this process is to produce Russian-English translations suitable for the analyses described in the following section.

1. Machine Translation

The 704 computer program for MT will eventually determine the structure of Russian sentences and construct equivalent English sentences. The program is expanded and improved as cycles of research produce more information about language, so it is impossible to give a final description of it. During the first cycle, the "machine-translation" program consisted solely of transliteration of the text and print-out of the glossary information. Analyses in the first cycle have led to the following machine routines, completed or planned:

a) Recognition of Idioms that Have Previously Occurred. An idiom is a sequence of forms that must be translated as a group, not one-by-one. This routine is ready for the second cycle.

b) Inflection of Nouns into Plural Number. The English equivalents in the glossary are generally uninflected. Hence it is necessary, when a Russian noun occurs in plural number, to inflect its English equivalent into the plural. A fairly complete routine is ready for the second cycle, but it does not take into account the fact that some forms of Russian nouns are ambiguous with respect to number. Extensions of the routine are planned to be in operation in the second cycle; these will use adjective-noun agreement to reduce the ambiguities.

c) Inflection of Verbs by Voice, Mood, Tense, Person, and Number. In English the inflection of verbs is more complicated than that of nouns. The third-person singular present tense, the past tense, the present participle, and the past participle require inflections; at times, auxiliary verbs and pronoun subjects also must be inserted. A routine to handle many inflections is planned to be in operation in the second cycle, but insertion of pronoun subjects in particular must wait for further textual analysis.

d) Insertion of Prepositions. When a Russian noun occurs in the genitive, dative, or accusative case, its English equivalent must, in most instances, be preceded by a preposition. The Russian noun may or may not be preceded by a preposition. A routine is planned to be in operation during the second cycle, which will connect Russian prepositions with their noun objects and will supply additional prepositions in English as required.

e) Selection of English Equivalents for Russian Prepositions. Russian prepositions have many alternative English equivalents. K. E. Harper, using the postedited corpus from the first cycle, has developed a classification of nouns that improves the accuracy of preposition translation. A routine is planned to be in operation during the second cycle, to select an equivalent for each preposition according to the class of the noun to which it is connected.

The computer program for machine translation has thus advanced since the first cycle began, but must be improved in every respect before machine translation is satisfactory without postediting.

The machine-translation stage concludes with the printing of a text list. The following items are printed in parallel columns:

> Sequence number — Coding space —
> Russian form — Grammar code —
> Primary English equivalent —
> Alternative English equivalents

The primary English equivalent, copied from the glossary in the first cycle, is to be modified by the machine-translation program in subsequent cycles.

The text list is designed to serve three different functions; its format economically provides for the support of these tasks:

(1) Evaluation of the Machine-translation Program. The quality of the program can be judged by reading the primary English equivalent column.

(2) Postediting. The posteditor, who must know both English grammar and the subject matter of the article can work from the English equivalents and the grammar code; he has no occasion to refer to the glossary. His notations are marked directly in the coding space; the text list then serves as a keypunch manuscript.

(3) Linguistic Analyses. The same list can be used by a linguist for structural or other analyses of the text.

2. Postediting

The posteditor inserts whatever notations are required to convert the rough machine translation into good English; his notations are analyzed in order to improve the glossary and the computer program. It is thus necessary for him to have good command of English grammar and the technical vocabulary of the scientific articles being translated. His task is to complete the work of the machine, so the rules

he follows must change from cycle to cycle as the machine-translation program develops. The following rules apply in the second cycle:

a) English Equivalents. The primary English equivalent is generally acceptable (see the following section, Glossary Refinement); if it is not, the posteditor makes one of three notations:

(1) He writes the code number of a listed alternative English equivalent in the coding space.
(2) He writes a new alternative English equivalent in the coding space.
(3) He writes a special symbol to denote that a string of occurrences is an idiom.

In one of these ways, the posteditor makes sure that the selected English equivalent is always acceptable in the context.

b) English Sentence Structure. The structure of the sentence is partially converted to English style by the machine-translation program; as that program develops in repeated cycles of research, fewer and fewer structural notes have to be made by the posteditor. Among his tasks are these:

(1) Inflection of English equivalents, or correction of the inflections made by the machine program.
(2) Insertion of English preposition codes when necessary, or correction of insertions made by the machine program.
(3) Insertion of codes giving correct English word order.

By such notations as these, the posteditor guarantees that the final product is grammatically acceptable in English.

c) Russian Sentence Structure. The posteditor indicates the connections in the sentence that make up its structure. Using such rules as the following, he writes next to each occurrence the sequence number of the occurrence on which it depends:

(1) Adjectives depend on the nouns they modify.
(2) Nouns that serve as objects of prepositions depend on the prepositions.
(3) Nouns that serve as subjects or objects of the verbs depend on the verbs.
(4) Words connected by conjunctions depend on the conjunctions.

The posteditor continues until every occurrence in the sentence, except one, is shown to depend on some other.

The selection of English equivalents and synthesis of English sentence structure was performed by the posteditor in the first cycle. Machine determination of Russian sentence structure is being initiated for the second cycle. The current rules for postediting are contained in MT Study 8.[6]

Analysis

The final component of this research methodology is analysis of the postedited translation, with the goal of refining both the glossary and the computer program. Some analyses are performed at the conclusion of each cycle; the advantages of this method include the following:

a) Compared with the preparation of a "complete" MT program before examination of any corpus, this method is more closely governed by the realities of language.

b) Compared with the translation of a very large corpus before any analysis or programming, this method is less costly, since it makes more efficient use of the posteditor's time. It is possible, by means of analyses in early cycles, to shift part of the work of corpus preparation from the editor to the computer program in subsequent cycles.

It follows that the two chief criteria for selection of analyses in each cycle are rapid reduction of the posteditor's work and selection of a corpus for each analysis large enough for statistical stability. Language problems that most often arise tend to satisfy both criteria in early cycles.

The method of analysis is empirical correlation of the posteditor's notations with the information in the glossary — word number, grammar code, and so forth. The following paragraphs describe some applications of the method.

1. Glossary Refinement

In each cycle, the glossary is enlarged by the addition of new forms and new idioms. In addition, analysis leads to improvement of the English equivalents. It is first necessary to determine, for each Russian word (i.e., set of forms) the minimal set of English equivalents required. The determination is made in the following steps:

a) A count is made of the number of occurrences for which each alternative equivalent is

6. H.P.Edmundson, K.E.Harper, D.G.Hays, "Studies in Machine Translation—8: Manual for Postediting Russian Scientific Text," in preparation.

preferred by the posteditor. The alternatives are rearranged in the glossary in order of frequency of preference.

b) In subsequent cycles, the posteditor is instructed to accept the first alternative as often as possible.

c) Secondary alternatives that are not preferred in subsequent cycles are deleted.

The English equivalents that remain are essential for accurate translation; thus it is necessary to develop criteria for choice of one of them in each context. The first task is to differentiate between the contexts in which a multiple-equivalent word is translated in different ways. The analytic text deck contains one card for every occurrence, and, alter postediting, each card is punched to show the English equivalent, and the words in the context summarized and tabulated. Presumably there are words that occur more often in the context of one preference than of the others; if such words exist, they permit differentiation of the contexts.

At least two more cycles are required before the RAND corpus will be large enough for this type of analysis. If, at that time, the data show strong differentiation of contexts, it will be necessary to construct models. One model that has been suggested is a thesaurus, or hierarchical classification of words. A model for semantic relations and a practical method for applying it are among the most important unsolved questions tions in the field of machine translation.

2. Computer-program Refinement

The general nature of the computer program is sketched in the previous section (Machine Translation). It consists of routines for determination of Russian sentence structure and construction of English sentences with equivalent structure. In early cycles, these tasks are performed by the posteditor; the purpose of analysis is to relate the actions of the posteditor to the observable characteristics of the Russian sentences, so that the computer can be programmed to take similar actions under similar circumstances.

Sentence structure is symbolized, in Russian and in English, by the following observable characteristics: word order, particles, inflections, agreements, and punctuation. For automatic computation, these characteristics are represented by word number, sequence number, grammar code, and punctuation code. Analysis consists of correlation of these characteristics of the Russian sentence with the English structural codes or structural-connection codes inserted by the posteditor.

The technique is to bring together all occurrences of form with a given grammar code — for example, all nouns in the dative plural. The analyst first tests whether any English structural code applies to all occurrences. For example, the English equivalents of Russian plural nouns must be inflected into the plural. A routine is established for English plural inflection, initiated when the Russian grammar code indicates a plural noun. Such grammatically determined routines are important, but they are few in number.

The next stage of analysis uses context of occurrence; all occurrences with a given grammar code are collected, and sorted according to grammar codes of contiguous forms. Taking the traditional rules of syntax as a guide, the analyst relates the English structural code to features of the context. The insertion of a preposition before the English equivalent of a Russian dative noun is thus related to the grammar codes of preceding occurrences. If the immediately preceding occurrence in Russian is a preposition, no additional preposition is required in English. Gradually extending the analysis over a wider context, the analyst connects dative plural nouns with preceding adjectives, preceding participial phrases, and prepositions preceding these modifiers. Syntactically determined computer routines for making the connections are written. The analyst is able to conclude that a dative noun, not connected with a preceding preposition, must be preceded by "to" in English translation. *

There are two limitations on this type of analysis. First, the structure of the sentence may be ambiguous; an adjective may be placed between two nouns with which it agrees — in Russian, it might modify either of them. It seems probable that true structural ambiguity is rare and that in most cases a sufficiently complex routine can resolve apparent ambiguities. The second limitation is that the routines are complicated by rules that are necessary for the resolution of extremely rare constructions. Since the routines must be stored in a computer of limited size, it is not practical to seek "perfect" machine translation.

---

* The example is taken from a study being conducted by D.G.Hays.

The analytic method described above is partially automatic; collection of occurrences with a given Russian grammar code, a given context, and a given English structural code is carried out by machine. With the explicit marking of structural connections planned for the second cycle, still more of the research operation becomes automatic, since it will be possible automatically to collect, for example, all dative plural nouns depending on prepositions, and to list all constructions that intervene between the preposition and the noun.

Conclusion

The RAND methodology is a system for preparing Russian scientific text on punched cards, for producing translations in analyzable form, and for exposing the relationships between the original and translated versions, semi-automatically, in such a way that translation can be programmed.

The research methodology described is, of course, designed to achieve satisfactory machine translation; the intermediate products are:

a) A descriptive grammar of the Russian language, as it is used today in scientific writing.

b) A working glossary of Scientific Russian with the English equivalents required for accurate translation.

Solutions to both conceptual and technical problems of computer application in linguistic research are given in the other papers of this series.

The Use of Punctuation Patterns in Machine Translation†

Gerard Salton, Computation Laboratory, Harvard University, Cambridge, Massachusetts

The determination of sentence structure contributes greatly to the understanding of written texts, and represents, therefore, an element of considerable value in mechanical translation. The present study deals with the analysis of English language punctuation patterns and presents a sample program for an automatic punctuation analysis.

SINCE A MACHINE can easily recognize punctuation marks, it is pertinent to inquire how the pattern of punctuation marks within a sentence can be used to determine sentence structure. Such a study may result in the development of criteria which can be incorporated in a program for machine translation. While the present study deals with the analysis of English language punctuation patterns, a similar analysis can of course be made for the punctuation of any other language.

Nature of Punctuation

All languages exhibit certain basic differences among the sentences or clauses which may be constructed. These differences are intuitively recognized, even by children who know, for example, that a statement requires no response, while a question requires an oral response. Thus, sentences may be classified as declarative, imperative, interrogatory, or exclamatory. Another classification might distinguish among clauses[1] simple, compound, or complex. One can intuitively postulate that punctuation marks are used in the written language to make these distinctions. Moreover, punctuation marks are also used to prevent ambiguity. Punctuation, then, is predominantly constructional, or grammatical, or logical. Because of this, certain invariances of punctuation patterns must exist. Indeed, most people will adhere to certain conventions and will strive to reduce ambiguity by the proper placement of punctuation within the sentences. In this sense, punctuation is part of the mechanics of writing, and insofar as it is, it will present a minimum of difficulties in the proposed analysis

On the other hand, punctuation also performs a non-logical, non-grammatical function, analogous to the part played in speech by intonation and pause and in writing by emphasis. In this respect, punctuation is an art rather than a part of the mechanics of writing. Clearly, this latter aspect makes it difficult to take certain patterns of punctuation marks and to draw conclusions from them about the sentence structure of the English language.

Fortunately, the situation may not be so troublesome as it might appear at first glance. Computing machines are not likely to be used to translate novels, poems and similar types of literary texts, but will probably be restricted to the translation of scientific texts. In view of this, a similar restriction can be imposed on the present analysis eliminating the more obvious cases of "psychological punctuation." Indeed, the more technical a text becomes, the

---

† This study was prepared during the Seminar in Mathematical Linguistics held at Harvard University in the spring of 1956. The writer is indebted to Dr. A. G. Oettinger and Mr. W. L. Eastman for helpful advice.

1. Unless otherwise noted, technical terms of grammar or syntax will always be used in the conventional sense as defined in elementary textbooks.

more predominant the logical and constructional element in the punctuation. Deviations from the standard punctuation patterns should be only a minor consideration for technical texts.

Choice of Punctuation Marks

The present analysis will include all those punctuational patterns which are commonly found in normal technical language. Included marks are the period, comma, semi-colon, colon, question mark, exclamation point, quotation mark, parenthesis, and dash. Excluded will be the hyphen, apostrophe, asterisk, multiple dashes, dots, compound points (such as comma dash, colon dash, etc.), and oblique stroke. These punctuation marks are not included in the present analysis because their influence on the sentence structure of scientific texts is believed to be small, either because of the special nature of these punctuation marks, or because of their comparative infrequency in scientific texts.

Among the punctuation marks to be considered, such as colon, semi-colon, and so on, certain occurrences or patterns which are theoretically possible may also safely be disregarded. Examples are given below:

(1) A colon is sometimes used to denote interpolation, as in the following sentence:

"He was good: he himself thought he was very good: at extricating himself from difficult situations."

(2) A semi-colon is sometimes used to separate clauses or phrases having common dependence as in the following sentence:

"There is tears for his love; joy for his fortune; honor for his valor; and death for his ambition."

Such uses are not expected to occur in texts which are likely to be analyzed by machine.

There is another class of occurrences of punctuation marks which, although not infrequent in technical texts, must be disregarded in the analysis. This class includes all non-punctuational uses such as occur in mathematical expressions. These can be eliminated without excessive difficulty when the text is being transcribed for input into the machine by enclosing anything resembling a mathematical expression or equation between a pair of asterisks. Punctuation marks which appear between asterisks are then disregarded automatically during the analysis.

The major classes of other non-punctuational uses to be eliminated are given below:

(1) Periods are used following sets of capital letters which stand for names or titles. For example, M.P. stands for Member of Parliament, or N.Y. stands for New York.

(2) Periods are used following abbreviations or contractions, as in Dr., Mr., Maj. Gen., etc., i.e., and so on. These uses are fairly frequent in scientific texts.

(3) Periods are used in section or chapter headings, as in "1.1. Communication. "

(4) Periods, commas, colons may also be used between digits in various capacities. Consider, for example, the following:

7:14:30 standing for 7 hours, 14 minutes, 30 seconds
12.50 standing for 12 1/2
12, 520 standing for 12 thousand 5 hundred and 20.

No general and attractive rule has been found for eliminating all of these situations satisfactorily. For case (1) it may be stipulated that the period following a single capital letter will be considered legitimate. If the period is followed by another capital letter and another period, then a special glossary must be consulted, and the periods must be disregarded. If the capital letters represent initials of proper names, they could not be found in the glossary; they would, however, have been eliminated from consideration by being enclosed between asterisks, as previously explained. A glossary must also be set up for the abbreviations mentioned under (2) above. Such glossaries should be fairly inclusive; in their construction, much help can be obtained from textbooks on punctuation, which usually contain long lists of these special words.

The punctuation marks in (3) and (4) can be eliminated by classifying them as arithmetic uses of punctuation. An expression such as 7:14:30 is then transmitted as *7:14:30*.

Additional work remains to be done if all non-punctuational uses of punctuation marks are to be eliminated. The rules specified here are sufficient, however, to dispose of the more troublesome cases.

Punctuation and Sentence Structure

Conventional textbooks on grammar generally classify punctuation marks in accordance with their function within a given sentence. Unfortunately, such a classification is difficult to

carry out by machine, since it requires the determination of syntactic or semantic characteristics of the related clauses or phrases. For this reason, it is convenient to let the punctuation patterns themselves serve as primary basis for classification in this study. [2] The classification is then independent of any relation which might exist between the punctuation and the surrounding text.

A sentence will be defined as a string of words between two punctuation marks of a certain type, and the parts of the sentence will be distinguished by their occurrences between two commas, or between a comma and a period, and so on. No consideration will be given to semantic characteristics, and syntactical relations will be used only when easily determinable. Instead, each word string between two punctuation marks will be treated as consisting of words belonging to specific word classes. These word classes will then be used in conjunction with the punctuation marks for analyzing the sentence. Usually a unique determination of the word class to which a word belongs will not be required; it is usually sufficient to store next to each word in the dictionary a code indicating the applicable word class or classes — noun, verb, adjective, etc. For present purposes the word classes are the conventional "parts of speech" of elementary grammar.

Proposed Analysis of Scientific Texts

Since the analysis of punctuation is to be done with the help of a computing machine, it is important to note a basic difference between the application of such a machine to the analysis of sentence structure, and the application of a computer to the solution of mathematical problems. When solving a mathematical problem, it is imperative that all possible special cases and exceptions be properly taken into consideration. A machine program which handles 95% of the cases which arise in the solution of a mathematical problem is not, in general, more useful than one which takes care of only 10% of the cases since neither program can be used to solve the problem. When analyzing sentence structure with the objective of improving word-for-word translation, the situation is quite different. Indeed, a program which could determine certain structural elements of 95% of the sentences analyzed would be very valuable. The 5% whose structure could not be successfully analyzed by the program could still be treated by word-for-word translation, so long as all the words were stored in the dictionary. The present analysis was made with this philosophy in mind, and not with the intention of achieving perfect sentence-for-sentence translation.

It has been suggested already, that the technical texts to be analyzed exhibit certain very special characteristics, so far as punctuation is concerned. The patterns of punctuation found in the texts that were considered during this study were invariably of a simple nature. A text of ten pages which was examined in detail was found to contain only two semi-colons, a few parentheses, and a few dashes, in addition to the liberal use of commas and periods. The sentences were comparatively long, however, averaging over twenty-five words per sentence. In view of the comparative length of the sentences, the main objective has been to reduce the size of the string of words to be translated as a unit by taking into account the punctuation pattern occurring in the sentence. This reduction of each sentence into parts or substrings was accomplished by specifying criteria to indicate which words belonged to the same substring, and which did not.

Two main criteria were used for reducing a sentence into substrings: for certain strings of words, the initial and final punctuation marks were found to be sufficient; for the remaining strings it was necessary to determine the word class of the various words between two successive punctuation marks. Wherever possible, exact determination of word classes was avoided by basing the rules on more general criteria. Thus, if a substring did or did not contain a verb, or if it did or did not contain a present participle; it was often possible to make a decision without the necessity of determining uniquely the word class of each word in that substring.

In order to reduce possible ambiguities to a minimum, the use of a few lists of words was introduced. The lists were always kept small, so that they could be searched quickly and would use little storage space. For example, a list was made of the most frequent words of inclusion. If a word appeared on the list, it was

2. An attempt to use punctuation marks for the determination of certain types of grammatical clauses was made by V. A. Oswald, Jr., and Stuart L.. Fletcher, Jr., "Proposals for the Mechanical Resolution of German Syntax Patterns, "Modern Language Forum 36, No. 3-4, 1951.

treated in some specified manner, without the necessity of determining its word class. It was decided that any remaining ambiguities would be ignored by the machine.

The reduction of the length of the strings of words was achieved by two basic operations: separation and compression. The process of separation may be described by noting that under certain circumstances a string of words, S, can be broken down into substrings S_1, S_2, . . ., S_n, which may be treated separately so far as the translation process is concerned. Such substrings might be separate clauses within a given sentence, or parenthetical thoughts, or any other parts of the sentence which may be treated apart from the remaining words. A transformation T may then be defined, which transforms the sentence S into its transform T(S). If T is translation, the T(S) stands for a new sentence in the output language. In general, the transformation T is not a uniquely defined operation and there is no reason why the same T should apply to all the substrings S_i.

There are, however, circumstances where T consists of exactly the same operations for a set of substrings in a given sentence. Under these special circumstances the second basic process, compression, can be used. To apply compression to a set of substrings, it is first necessary to identify the substrings by noting the similarity in their formal structure; this, in turn, requires an exact determination of the word classes. However, once the substrings have been identified, the operator T need be determined for only one of the substrings, and can then be applied automatically to the others. Elements in an enumeration or a series, usually separated by commas in the sentence, are substrings of this type. Consider, as an example, the following string of words:

> "Here we have a peach, an orange, an apple, a pear and thirty, perhaps thirty-one, grapes."

It may be noted first that the words "a peach, an orange, an apple, a pear" constitute a set of four indefinite articles each followed by a noun which is separated from the next article by a comma. Clearly, four substrings may be recognized, each consisting of one indefinite article and the noun immediately following it. As soon as a set of operations, T_i, has been specified which may be applied to one of the substrings, the same T_i may be applied to the other three substrings. It may be noted next that the words "perhaps thirty-one" between the remaining two commas really express a parenthetical thought, which may be treated separately from the remaining parts of the sentence. The sentence may now be written as follows:

Here we have { a peach / an orange / an apple / a pear } and thirty

(perhaps thirty-one) grapes.

The punctuation, in this instance a set of five commas, was used to "separate" the parenthesis as indicated by a loop, and to "compress" the series as indicated by the brackets. In addition to the punctuation, the pattern of word classes in the sentence was also taken into account to assist in the appropriate classification of the substrings.

A set of specifications for a machine program is presented below. It is believed that any moderately experienced programmer can easily write a program for these specifications. An actual program has not been written, because it is felt that some of the rules could be improved, after further analysis. Specifically, some of the word lists require more extensive research. However, it is believed that the specifications as they stand, are capable of treating adequately about 95% of the word strings likely to be encountered in technical texts, and that they contain the main bulk of the results which can be hoped to be achieved by punctuation anal-
ysis. This, of course, requires experimental confirmation. No machine analysis was made. The sample text reproduced at the end of this article was analyzed by hand, care being taken to proceed with the analysis in the same step-by-step manner which would have been used, had the analysis been carried out by machine.

Program Specifications

(A) Step One

Consider one sentence at a time, that is, one string of words included between any two of the following: exclamation point, question mark, period. Mark the beginning and end of the sentence by two strokes and eliminate the corresponding punctuation mark.

Of course, before being eliminated, any punctuation mark must first be tested as to whether it might be used in some non-punctuational manner. If this is the case, it is merely passed over, and the next punctuation mark is considered.

(B) Step Two
Within every sentence mark the position of any pair of parentheses, dashes, or quotation marks by square brackets. When a dash (or parenthesis) is followed by another dash (or parenthesis) in the same sentence, the two will be considered a pair. Four dashes in the same sentence will be considered two pairs, and so on.

(C) Step Three
Mark the position of any semi-colon, colon, or single dash in the sentence by two strokes, except when these punctuation marks occur within a pair of parentheses, quotation marks, or dashes, in which case mark their position by one stroke. The strokes again replace the punctuation.

(D) Step Four
Consider the occurrence of commas within adjoining two-stroke marks; commas which are included in the brackets which have been set up in Step Two must be considered separately from those commas not within the brackets. If a series is detected, compress the terms of the series and mark by braces as shown in the example previously given.

Series use of the comma is defined as the occurrence in any sentence, of two or more terms of similar grammatical construction and common dependence, the terms of the series being separated by a comma. In a series of more than two terms, the last comma may or may not be followed by a conjunction. The terms of the series may be nouns, pronouns, adjectives, verbs, adverbs, prepositions, or combinations of the foregoing.

Some series will not be recognized because of small deviations in word patterns. Such substrings will be treated as parenthetical in Step Five without substantial loss in meaning. Examples are given below:

> Good jobs are now plentiful, construction activity high, and employment at its peak.

Here the words "are now" or "is now" are not repeated in the second and third term of the series. These will, however, be treated as parenthetical by Step Five.

> The strengthening of technical, college level, and advanced education is of paramount importance to the United States.

Here "college level" is used as an adjective and will not be recognized as such by the machine. The term will again be treated as parenthetical.

(E) Step Five
Consider the remaining commas within adjoining two-stroke marks, and mark by a loop any substring which may be considered a parenthesis and may therefore be treated without reference to the remaining words within the two-stroke marks. A parenthesis may be placed at the beginning of the string within adjoining two-Stroke marks, if it starts with a two-stroke mark and ends with a comma; in the middle of the string if it starts with a comma and ends with a comma; and at the end of the string if it starts with a comma and ends with a two-stroke mark. Commas occurring within square brackets must be treated separately from those outside the square brackets. The parentheses will be marked in the following sequence:

(a) strings of words in the middle (that is, both preceded and followed by a comma) not containing any verb

(b) strings of words in the middle containing no verb form except a present participle (a verb form ending in "ing")

(c) strings of words in the middle containing no verb form except a past participle. (If past participles are not stored explicitly in the machine, this step may be omitted.)

(d) strings of words as defined in (a), (b), or (c) if they occur at the beginning or at the end of the substrings instead of occurring in the middle, except that any string of words at the beginning or at the end which adjoins a string already looped will not be looped, since it is already separated from the remainder of the substring

(e) strings of words which contain one of the verbs used to interrupt a narrative or quotation (say, tell, observe, suppose, interrupt, etc.) accompanied by any of the following: pronoun, noun, article, adjective, adverb, or any combination thereof. (This category is not believed important in technical texts and could be ignored.)

(f) strings of words containing no verb form other than an infinitive

(g) strings of words in the middle of a substring, starting with an adverbial clause or phrase (in order to do this, on the same day, etc.). If an exhaustive list of such adverbial clauses is not available, this category could also be ignored.

(h) strings of words in the middle starting with any of the following words of inclusion (words which refer to clauses after them):

after	since	wherever
although	that	whether
as	unless	which
because	what	whichever
before	whatever	while
how	when	who
if	whenever	whoever
in order that	where	why

(i) strings of words in the middle containing in juxtaposition two conjunctions, or adverb and conjunction, or conjunction and adverb (these are always subordinate clauses of some sort). Some of the more frequent occurrences are: whether if, that since, since even, then if, that if, even if, even while, even though, etc.

(F) Step Six

Consider the remaining commas, within adjoining two-stroke marks, (that is, those not already eliminated by brackets, braces, or loops) and mark the position of the comma by one stroke in the following order:

(a) strings of words starting with any simple coordinating conjunction (and, or, but, for, nor, neither)

(b) strings of words which fall under (e), (f), or (g) of Step Five, but which are at the beginning or at the end of the substring

(c) any comma not already eliminated should be replaced by one stroke.

In connection with Steps Five and Six, it should be mentioned that the machine program associated with each of the above steps is obvious from the description of the steps themselves, since these are based either on lists of words which must be searched or on certain easily recognizable word forms or classes.

Residual Problems

The program specifications outlined in the preceding section will properly recognize most punctuation patterns. There are, however, certain exceptions which are not covered as yet. Some examples are given below:

1. How could a comma used to replace missing words be properly recognized? For example, in the sentence

"I am fond of apples; he, of pears."

"he" and "of pears" would be recognized as parenthetical by Step Five.

2. How can series commas be recognized? Consider for example:

"The stooped, meticulously clad figure... "

Here "stooped" and "meticulously clad" ought to be recognized as two adjectives.

3. How can inversions be recognized? Consider

"..., the treatment of which may be stopped..."

Here the preposition and conjunction do not immediately follow the comma because of the inversion; however, the sentence ought to be treated as if preposition and conjunction did introduce the subordinate clause.

While these and other cases may not be included in the program specifications, the efficiency of the punctuation analysis is not greatly reduced, since constructions of this kind are relatively rare.

Conclusion

A text is analyzed in the next section to show the possibilities inherent in the proposed method. The sentences were almost invariably analyzed correctly. For some sentences no additional information was furnished by the analysis; for others, a correct analysis was dependent either on the correct determination of word classes or on the recognition of abbreviations and other non-punctuational uses of punctuation.

In spite of the difficulties, it is believed that the modest attempt made here at analyzing the structure of punctuation has been worth while. If a sufficient number of structural elements in the language are analyzed, and rules are specified for inclusion in an automatic translation program, it may eventually be possible to attain the accuracy of sentence-by-sentence translation.

Sample of Simulated Automatic Punctuation Analysis

The paragraph chosen for analysis is not from a scientific text but rather reproduces a conversation. It was chosen because no scientific text could be found which would exhibit a large enough variety of punctuation within a short paragraph to illustrate many of the steps described in the previous section. An immediate consequence of the non-technical character of the text is the fact that the sentences tend to be rather short, making for greater simplicity in the analysis. This should not, however, de-

tract from the fact that all the sentences in this text could be analyzed properly, despite a great variety of punctuational patterns. In further experiments with technical material long sentences were successfully analyzed.

The text is first given unaltered. Thereafter the text is reproduced after each of the six steps. Wherever the number of changes for any one step was very small, only the sentences which were actually changed are reproduced. The last copy of the text is broken down into substrings which start and end with a double stroke.

Text Before Simulated Automatic Analysis

Well, well, well! What are you* doing here, you old rascal?

Oh, nothing much.

Just looking around for what you can pick up, I suppose?

Tut, tut! Don't make me out to be a thief. I'm only an opportunist, after all, you know.

But a very unusual sort of opportunist: You don't merely grasp such opportunities as offer themselves to your remarkably observant eyes, Charles; you do a great deal — or so, at least, I suspect — to create the opportunities.

Why, Robert! The conversation is taking a strange turn. You began by almost insulting me; now you are paying me what is, in effect, a compliment — indeed, in its way, a very high compliment.

Darn it! That's not quite correct — I mean about the compliment — for although I admire your versatility, your resourcefulness, your adaptability —

Don't pile it on too thick, old man!

Don't interrupt! As I was saying, I admire these qualities; others I deplore. But you are quite right: we seem to be dropping into a fin de siecle persiflage and hatred of being earnest. However, I wanted to ask you for —

Sorry, old chap! Must tear myself away — my bus, you know.

Text After Step One

// Well, well, well // What are you doing here, you old rascal //

// Oh, nothing much //

// Just looking around for what you can pick up, I suppose //

// Tut, tut // Don't make me out to be a thief // I'm only an opportunist, after all, you know //

// But a very unusual sort of opportunist: You don't merely grasp such opportunities as offer themselves to your remarkably observant eyes, Charles; you do a great deal — or so, at least, I suspect — to create the opportunities //

// Why, Robert // The conversation is taking a strange turn // You began by almost insulting me; now you are paying me what is, in effect, a compliment — indeed, in its way, a very high compliment //

// Darn it // That's not quite correct — I mean about the compliment — for although I admire your versatility, your resourcefulness, your adaptability —

Don't pile it on too thick, old man //

// Don't interrupt // As I was saying, I admire these qualities; others I deplore // But you are quite right: we seem to be dropping into a fin de siecle persiflage and hatred of being earnest // However, I wanted to ask you for —

Sorry, old chap // Must tear myself away — my bus, you know //

Excerpt from Text after Step Two

//But a very unusual sort of opportunist: You don't merely grasp such opportunities as offer themselves to your remarkably observant eyes, Charles; you do a great deal [or so, at least, I suspect] to create the opportunities//

\- - -

//That's not quite correct [I mean about the compliment] for although I admire your versatility, your resourcefulness, your adaptability —

Don't pile it on too thick, old man//

Text after Step Three

//Well, well, well / What are you doing here, you old rascal//

//Oh, nothing much //

//Just looking around for what you can pick up, I suppose //

//Tut, tut //Don't make me out to be a thief// I'm only an opportunist, after all, you know //

//But a very unusual sort of opportunist // You don't merely grasp such opportunities as offer themselves to your remarkably observant eyes, Charles // you do a great deal for so, at least, I suspect] to create the opportunities //

//Why, Robert // The conversation is taking a strange turn // You began by almost insulting me // now you are paying me what is, in effect, a compliment // indeed, in its way, a very high compliment //

---

* The words which are underlined were italicized in the original text.

//Darn it// That's not quite correct [I mean about the compliment] for although I admire your versatility, your resourcefulness, your adaptability //

// Don't pile it on too thick, old man//

//Don't interrupt //As I was saying, I admire these qualities // others I deplore // But you are quite right // we seem to be dropping into a fin de siecle persiflage and hatred of being earnest // However, I wanted to ask you for //

// Sorry, old chap // Must tear myself away // my bus, you know//

Excerpts from Text after Step Four

// { Well / well / well } // What are you doing here, you old rascal //

\- - -

// Darn it // That's not quite correct [I mean about the compliment] for although I admire { your versatility / your resourcefulness / your adaptability } //

\- - -

Text after Step Five

// { Well / well / well } // What are you doing here (you old rascal) //

// (Oh) (nothing much) //

// Just looking around for what you can pick up, I suppose //

// { Tut / tut } // Don't make me out to be a thief // I'm only an opportunist (after all) you know //

// But a very unusual sort of opportunist // You don't merely grasp such opportunities as offer themselves to your remarkably observant eyes (Charles) // you do a great deal [or so (at least) I suspect] to create the opportunities //

// Why (Robert) // The conversation is taking a strange turn // You began by almost insulting me // now you are paying me what is (in effect) a compliment // indeed (in its way) a very high compliment //

// Darn it // That's not quite correct [I mean about the compliment] for although I admire { your versatility / your resourcefulness / your adaptability } //

// Don't pile it on too thick (old man) //

// Don't interrupt // As I was saying, I admire these qualities // others I deplore // But you are quite right // we seem to be dropping into a fin de siecle persiflage and hatred of being earnest // // (However) I wanted to ask you for //

// (Sorry) (old chap) // Must tear myself away // (my bus) you know

Excerpts from Text after Step Six

// Just looking around for what you can pick up / I suppose //

\- - -

// Don't interrupt // As I was saying / I admire these qualities // others I deplore //

Final Text No. 1

String No.

(1) { Well / well / well }

(2) What are you doing here (you old rascal)

(3) (Oh) (nothing much)

(4) Just looking around for what you can pick up / I suppose

(5) { Tut / tut }

(6) Don't make me out to be a thief

(7) I'm only an opportunist (after all) you know

(8) But a very unusual sort of opportunist

(9) You don't merely grasp such opportunities as offer themselves to your remarkably observant eyes (Charles)

(10) you do a great deal [or so (at least) I suspect] to create the opportunities

(11) Why (Robert)

(12) The conversation is taking a strange turn

(13) You began by almost insulting me

(14) Now you are paying me what is (in effect) a compliment

(15) indeed (in its way) a very high compliment

(16) Darn it

(17) That's not quite correct [I mean about the compliment] for although I admire
{ your resourcefulness
your versatility
your adaptability }

(18) Don't pile it on too thick (old man)

(19) Don't interrupt

(20) As I was saying / I admire these qualitie

(21) others I deplore

(22) But you are quite right

(23) we seem to be dropping into a fin de siecl persiflage and hatred of being earnest

(24) (However) I wanted to ask you for

(25) (Sorry) (old chap)

(26) Must tear myself away

(27) (my bus) you know

A Programming Language for Mechanical Translation†

Victor H. Yngve, Massachusetts Institute of Technology, Cambridge, Massachusetts

A notational system for use in writing translation routines and related programs is described. The system is specially designed to be convenient for the linguist so that he can do his own programming. Programs in this notation can be converted into computer programs automatically by the computer. This article presents complete instructions for using the notation and includes some illustrative programs.

IT HAS BEEN SAID that the automatic digital computer can do anything with symbols that we can tell it in detail how to do. If we are interested in telling a digital computer to translate texts from one language into another language, we are faced with two tasks. We first have to find out in detail how to translate a text from one language to another. Then we have to "tell" the computer how to do it. This paper is concerned with the second task. We will present here a specially devised language in which the linguist can conveniently "tell" the computer to do things that he wants it to do.

The automatic digital computer has been designed to handle mathematical problems. It is able to carry out complicated routines in terms of a few different kinds of elementary operations such as adding two numbers, subtracting a number from another number, moving a number from one location to another, taking its next instruction from one of two places depending on whether a given number is negative or positive, and so on. In order to instruct the computer to carry out complicated routines, simple instructions for the elementary operations are combined into a program. The writing of a program to carry out even an apparently rather simple procedure can be an exacting task requiring a high degree of skill on the part of the programmer.

It has been the custom for the linguist who wanted to try out a certain approach to mechanical translation to ask an expert programmer to program his material rather than to learn the art of programming himself. Besides the usual inconveniences and difficulties attending the communication between experts in two separate fields, this practice has certain more basic difficulties: Neither the linguist nor the programmer has been able to be fully effective. The linguist has not become aware of the full power of the machine, and the programmer, not being a linguist, has not been able to use his special knowledge of the machine with full effectiveness on linguistic problems.

The solution offered here to these difficulties is an automatic programming system. The linguist writes the results of his research in a notation or language called COMIT, which has been specially devised to fill his needs. The programmer writes a conversion program or compiler capable of converting anything written in this notation into a program that can be run on the computer.* Thus the expense, time, and effort needed to separately program each linguistic approach is saved, and, even more important, the linguist is given direct access to the machine. He becomes more fully aware of its potentialities, and his research is greatly facilitated.

† This work was supported in part by the U. S. Army (Signal Corps), the U. S. Air Force (Office of Scientific Research, Air Research and Development Command), and the U.S.Navy (Office of Naval Research); and in part by the National Science Foundation.

* This is being done by the programming research staff of the M.I. T. Computation Center.

What COMIT Is

COMIT is an automatic programming system for an electronic digital computer that provides the linguist with a simple language in which he can express the results of his researches and in which he can direct the computer to analyze, synthesize, or translate sentences. It is capable of being programmed on any general purpose computer having enough storage and appropriate input and output equipment. The language has been devised to meet the needs of the linguist who wants to work in the fields of syntax and mechanical translation. Some of the linguistic devices and operations that COMIT has been designed to express are: immediate constituent structure, discontinuous constituents, coordination, subordination, transformations and rearrangements, change in the number of sentences or clauses in translation, agreement, government, selectional restrictions, recursive rules, etc.

A program written in COMIT consists of a number of rules written in a special notation. The computer executes these rules one at a time in a predetermined order. In seeking an appropriate notation in which to write the rules, we were guided by several considerations.

1. That the rules be convenient for the linguist - compact, easy to use, and easy to think in terms of.
2. That the rules be flexible and powerful — that they not only reflect the current linguistic views on what grammar rules are, but also that they be easily adaptable to other linguistic views,

A linguist can use the computer in the following simple way. He expresses the results of his linguistic research in COMIT. He transcribes his rules onto punched cards using a device with a typewriter keyboard. He supplies text or special instructions to the machine also on punched cards. He then gives these packs of cards to an operator and subsequently receives his results in the form of printed sheets from the machine.

The way that a COMIT program works in the computer is shown in figure 1. The rules making up the COMIT program can be thought of as stored in the computer at A. Material to be translated or otherwise operated on enters the computer under the control of the rules from the input B. It is operated on by the rules and translated in the workspace C. It then goes to the output E. The dispatcher D contains special information, stored there by the rules, which governs the flow of control or the order in which the rules of the program are carried out.

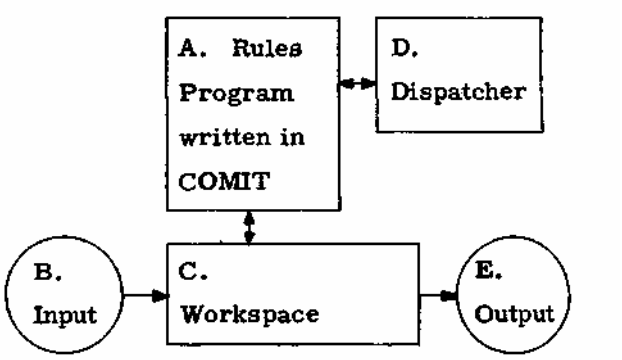


Fig. 1. How a COMIT program works in the computer.

The way in which COMIT rules are written, how they direct the computer to perform the desired operations, and how they are assembled into programs will now be described. The remainder of the paper is thus a complete manual of detailed instructions for using this special-purpose programming language.

COMIT Rules and Their Interpretation

A rule in COMIT has five sections, the name, the left half, the right half, the routing, and the go-to, each with its special functions. Figure 2 shows how a rule is divided into these

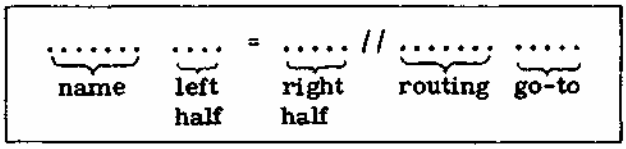


Fig. 2. The five sections of a rule in COMIT.

five sections. The name and left half are separated by a space, the left half and the right half are separated by an equal sign, the right half and the routing are separated by two fraction bars, and the routing and the go-to are separated by a space;

— flow of control —

We will discuss first the function of the name and the go-to, which have to do with the flow of control from one rule to another. A program written in COMIT always starts with the first rule in sequence. After a rule has been carried out, the computer obtains in the go-to the name of the next rule to be carried out. The name of each rule is to be found in the left-hand part of the name section of that rule. (The

right-hand part of the name section is reserved for the subrule name, to be discussed later.)

In addition there are three cases when control is automatically transferred to the next rule in sequence regardless of its name. One of these will be immediately clear; the other two will be clarified in the explanations of the left half and the routing. The three are: (1) an asterisk is written in the go-to, (2) the constituents written in the left half of the rule were not found in the workspace, (3) an *R in the routing finds no more material at the input. A rule to which control is always transferred automatically in this fashion so that a rule name is not needed, may have an asterisk in the name section in place of a rule name. When this automatic transfer of control takes place from the last rule in sequence so that there is no next rule, the COMIT program stops.

Figure 3 shows an example of how control proceeds from one rule to another under the direction of the rule name and the go-to sections. In this program, rule A would be the first one executed, then C, then the rule with an asterisk in the name section, then B, then C, then *, then back to B again, and so on round and round in what is known as a loop, until one of the conditions occurs in the rule marked asterisk that will automatically transfer control to the next rule D. After D has been executed, the program will stop.

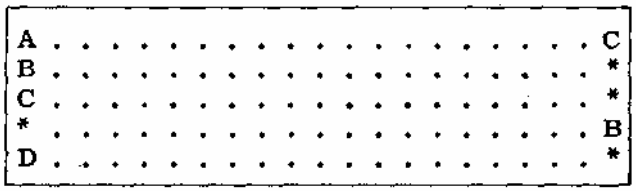


Fig. 3. A COMIT program to illustrate the flow of control under the direction of the rule name and the go-to sections of the rules.

As an aid to the memory, we will give a way in which each part of a rule in COMIT can be read in English. This will be done by providing English equivalents for all abbreviations used in COMIT, and by providing certain conventional wordings that will always be used between the various sections and between the various abbreviations. For the parts of the rule already discussed we need the following conventions: A rule is preceded by the word "in", rule names are preceded by the words "the rule", the go-to is preceded by the words "then go to", an * in the name section is read "this rule", an * in the go-to is read "the next rule, " and the rule is followed by a period to make a sentence. These conventions are enough to read the program in figure 3. These and the other conventions are conveniently tabulated in a later section. According to the conventions, the program in figure 3 should be read:

In/the rule A/... /then go to/the rule C/.
In/the rule B/... /then go to/the next rule/.
In/the rule C/... /then go to/the next rule/.
In/this rule /... /then go to/the rule B/.
In/the rule D/... /then go to/the next rule/.

The dispatcher also can influence the flow of control in the following way: A rule in COMIT may have several subrules. In figure 4, the rule B has four subrules. The rule name is

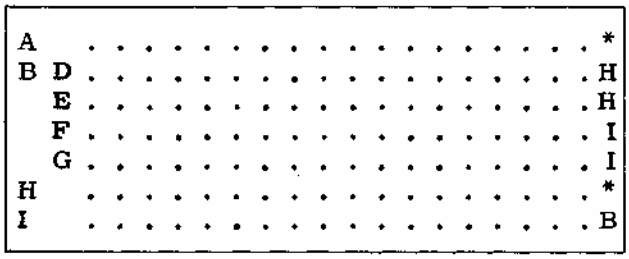


Fig. 4. A COMIT program to illustrate a rule with subrules. The rule B has four subrules.

in the left hand part of the name section of the first subrule. The name of each subrule is in the right hand part of the name section of that subrule. A rule that does not have several subrules may be thought of as a rule with just one subrule. A rule with only one subrule does not have a subrule name. When control is transferred to a rule with several subrules, the dispatcher is consulted for an indication of which subrule is to be carried out. For this purpose the dispatcher contains dispatcher entries. A dispatcher entry of the form B E would cause the computer to execute the subrule E in rule B each time it comes to that rule. If there is no entry in the dispatcher for this particular rule, or if there is an entry, but it contains more than one subrule name, the choice is made at random. In other words, if the dispatcher contains the entry B E G, the computer will choose at random between the two alternative subrules E and G. A dispatcher entry having a minus sign in front of its values (subrule names) has the same meaning as it would have if it had all its possible values except those following the minus sign. A dispatcher entry with a rule

name but no values has the same meaning as one with all possible values, that is, choose completely at random. The contents of the dispatcher are not altered by any of these processes. How the contents of the dispatcher may be altered will be discussed in the section on the routing.

The English reading of a rule with several subrules is the same as that for a rule with one subrule except that the words "consult the dispatcher and select" are read following the rule name. In figure 4, the rule B with four subrules is read:

In/the rule B/consult the dispatcher and select/
the subrule D/. . . /then go to/the rule H/.
the subrule E/... /then go to/the rule H/.
the subrule F/... /then go to/the rule I/,
the subrule G/... /then go to/the rule I/.

— workspace —

Having discussed the flow of control, we will turn to the workspace and describe how text to be translated or other material to be worked on is represented there. This will prepare us for a discussion of the remaining three parts of the rule whose function it is to operate on the material in the workspace.

Material is stored in the workspace as a series of constituents separated by plus signs. A constituent consists either of a symbol alone or a symbol and one or more subscripts. The symbol is written first. It may be the textual material itself, a word, phrase, or part of a word; or it may be any temporary word or abbreviation that the linguist finds convenient to use. Subscripts are of two kinds, logical subscripts and numerical subscripts. Logical subscripts are potential dispatcher entries and thus have the form of a rule name (subscript name) followed by one or more subrule names (values). Numerical subscripts are used for numbering and counting purposes. They consist of a period for the subscript name followed by an integer n in the range $0 \leq n < 2^{15}$. A constituent may have any number of logical subscripts, but only one numerical subscript.

An example of how linguistic material can be represented in the workspace is given in figure 5. This could be read in English as follows: "a constituent consisting of/the symbol IN/ with/the numerical subscript/1/ , followed by/ a constituent consisting of/the symbol DER/ with/the numerical subscript/2/ , followed by/ a constituent consisting of/the symbol ADJ/with/ the numerical subscript/3/ , and with/the subscript AFF/having/the value EN/ , followed by/a constituent consisting of/the symbol NOUN/ with/the numerical subscript/4/ , and with/the subscript GENDER/having/the value FEM/." The conventional wordings and the readings for the abbreviations used may be found tabulated near the end of this article.

```
. IN/ . 1 + DER/ . 2 + ADJ/ . 3, AFF EN
  + NOUN/ . 4, GENDER FEM
```

Fig. 5. Example of how linguistic material may be represented in the workspace.

- left half -

Having discussed the name and go-to sections and shown how material is represented in the workspace, we are now ready to discuss the remaining three sections of a rule. First we will take up the left half. A rule with several subrules may have no more than one left half. It is written in the first subrule. The function of the left half is to indicate to the computer which constituents in the workspace are to be operated on by the rest of the rule. The constituents in the workspace to be operated on are indicated by writing constituents in the left half that match them in certain definite respects.

A match condition between a constituent in the workspace and a constituent written in the left half will be recognized if the following conditions hold: (1) The symbols are identical. (2) If the constituent in the left half has any subscripts written on it, the constituent in the workspace must also have at least subscripts with the indicated subscript names — the order of writing the subscripts has no significance. (3) If the logical subscripts in the left half have any values indicated, the subscripts in the workspace must also have at least these values — again the order is unimportant. (4) If a numerical subscript is written in the left half, the numerical subscript in the workspace must have an identical numerical value, but if . G or . L is written in the left half before the value of a numerical subscript, a numerical subscript in the workspace will be matched if it has, respectively, a value greater than or less than the value written in the left half.

Dollar signs written in the left half have special meanings. $1 may be written in the left half to match any arbitrary symbol. If the $1 is followed by subscripts, they are matched in the normal fashion. A dollar sign followed by any number greater than 1 ($4) will match the

indicated number of constituents. It cannot have subscripts. A dollar sign without a number can be written as a constituent in the left half and can match any number of constituents in the workspace, including none. This is called an indefinite dollar sign, while those with numbers are called definite dollar signs.

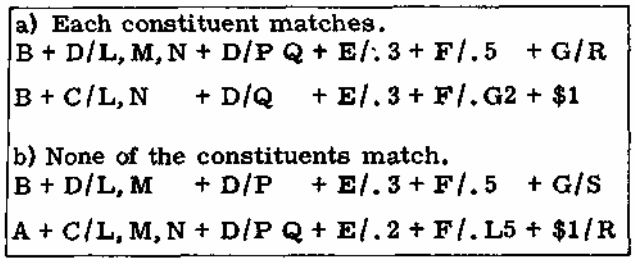


Fig. 6. Examples of match and no-match conditions. The top lines in a) and b) represent constituents in the workspace. The bottom lines represent constituents as written in the left half.

As an example of how constituents written in the left half can match constituents found in the workspace, figure 6 a shows several of the possibilities. Each constituent in the second line represents a constituent as it might be written in the left half. It matches the workspace constituent written directly above it in the first line. In figure 6 b, none of the constituents meet the match conditions.

The computer carries out a search for a match condition between each of the constituents written in the left half and corresponding constituents in the workspace in the following way: The first constituent on the left in the left half is compared in turn with each constituent in the workspace starting from the left until a match is found. The computer then attempts to match the next constituent in the left half with the next constituent in the workspace and so on until either all constituents written in the left half have been matched, or one constituent fails to match. In this case, the computer starts again with the first constituent in the left half and searches for another match in the workspace. Finally, either a match is found for all of the constituents and the computer goes on to execute the rest of the rule, or the computer cannot find the indicated structure in the workspace, in which case control is automatically transferred to the next rule. It can be seen that a structure will be found in the workspace only if it has matching constituents that are consecutive and in the same order as those written in the left half.

If an indefinite dollar sign is the first constituent in the left half, it will match all of the constituents in the workspace to the left of any constituent that is matched by the second constituent in the left half. If the indefinite dollar sign is the last constituent in the left half, it will match all of the constituents in the workspace to the right of any constituent that is matched by the next to the last constituent in the left half. If there are two or more indefinite dollar signs written in the same left half, they must be separated by constituents that are not dollar signs, or by $1 with subscripts, in order to prevent an ambiguity as to which constituents in the workspace are to be found by the several indefinite dollar signs.

If an indefinite dollar sign has constituents written on each side of it in the left half, the computer will first try to match all constituents to the left of the indefinite dollar sign. It does not have to search again for the constituents to the left of the dollar sign unless a number (as will be explained shortly) referring to a constituent to the left of the indefinite dollar sign is written to the right of the indefinite dollar sign. In this case, the computer will search for a new match for constituents to the left of the indefinite dollar sign if it fails to find a match with the constituents to the right of the indefinite dollar sign.

Constituents in the left half are conceived of as being numbered starting with one on the left. The leftmost constituent is called the number one constituent in the left half. When the constituents written in the left half have been successfully matched with constituents in the workspace, the constituents in the workspace that have been found are temporarily numbered by the computer in the same way as the constituents in the left half. The constituent in the workspace found by the number one constituent in the left half thus becomes the number one constituent in the workspace. The temporary numbering of constituents in the workspace remains until it is altered by the right half or until the rule has been completely executed. Its purpose is to allow expressions in the left half, right half and routing to refer to constituents in the workspace by their temporary number.

The various steps in a search are indicated in the example given in figure 7. The lower two lines give the constituents as they are written in the left half of a rule, and the way in

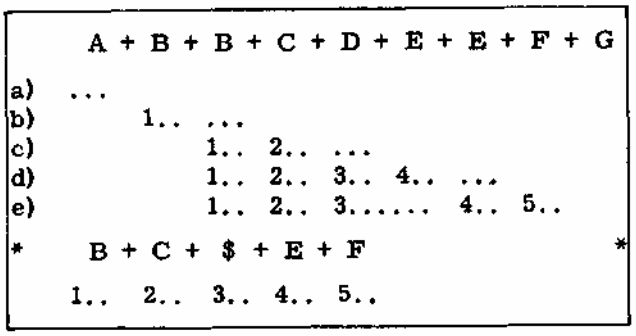


Fig. 7. Example of the search steps that the computer goes through in order to find in the workspace (top line) the structure written in the left half of the rule (next to bottom line).

which the computer numbers these constituents. The top line indicates the current contents of the workspace. Lines a) through e) represent the way in which the computer temporarily numbers the constituents in the workspace that have been successfully matched at each step of the search. The first step is indicated in line a): an attempted match between the number one constituent in the left half and the first constituent on the left in the workspace fails. In line b), the number one constituent matches the second constituent in the workspace, but an attempted match between the number two constituent in the left half and the third constituent in the workspace fails. In line c), the number one constituent in the left half matches the third constituent in the workspace, and the number two the fourth, but since the number three constituent is an indefinite dollar sign and can match any number of constituents including none, the next constituent, number four is matched with the fifth in the workspace. The match fails. Having already matched the constituents in the left half to the left of the indefinite dollar sign, the computer now tries to match the constituents to the right of the indefinite dollar sign. In line d), it finds a match of the number four constituent with the sixth, but the number five constituent in the left half fails to match the seventh constituent in the workspace. The computer then tries again with the number four constituent, and in e) finds a match between the number four and number five constituents in the left half and the seventh and eighth constituents in the workspace. Since all of the constituents in the left half have now been found in the workspace, the constituents in the workspace that have been found are left with the numbers as shown in line e). The third, fourth, fifth and sixth, seventh, and eighth constituents in the workspace become respectively the number one, two, three, four, and five constituents in the workspace. Note that two or more constituents in the workspace may be given one number if they are referred to by a dollar sign in the left half.

It is possible for the left half to be modified to some extent by what is found in the workspace . This can be done by writing a number as a constituent in the left half. The number then refers to the constituent already found in the workspace that has been given that number. The rest of the left half is then executed as if the constituent referred to in the workspace had been written originally in the left half in place of the number. A number written in the left half can only refer to a constituent in the workspace that has already been found by a constituent to the left of it in the left half. It can refer only to a single constituent, one matched by $1 for example. A number written in the left half cannot have subscripts written on it.

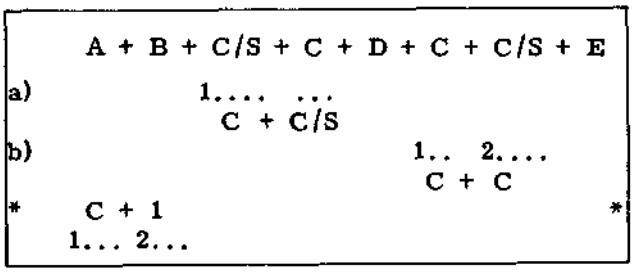


Fig. 8. Example of use of a number in the left half (bottom two lines). Attempted match indicated at a) fails, but the one at b) is successful. The contents of the workspace are represented on the top line.

Figure 8 gives an example of the use of a number in the left half. After two unsuccessful matches, the number one constituent in the left half finds the third constituent in the workspace. The number two constituent in the left half is then considered to be replaced by this constituent that has just been found (C/S). The match then fails because the fourth constituent in the workspace does not have at least the subscript S, required for a match condition. But when the number one constituent in the left half finally finds the sixth constituent in the workspace, the number two constituent in the left half is considered to be replaced by this constituent (C), and the next match is successful because this C will, according to the conditions for a match, find the C/S that is next in the workspace.

The English reading of the left half is the same as the reading of the material in the workspace except that it starts with ", search for a match in the workspace for", ends with ",and if not found, go to the next rule, but if found ", and includes conventional wordings for several abbreviations including the dollar signs and the numbers. For example, A/.G3 + $1 + $ + $2 + 2 in the left half would be read: ", search for a match in the workspace for /a constituent consisting of /the symbol A/with/the numerical subscript/greater than/3/, followed by/a constituent consisting of/any symbol/, followed by /a constituent consisting of/any number of constituents/, followed by/a constituent consisting of/two constituents/, followed by/a constituent consisting of/the number two constituent in the workspace /, and if not found, go to the next rule, but if found".

- right half -

The function of the right half is to indicate how the structures found in the workspace by the left half are to be altered. If there is no right half, the structures found in the workspace are left unaltered.

Rearrangement of the constituents found by the left half and temporarily numbered will take place when the appropriate numbers are written in the right half in the desired new order. If any of the numbers referring to constituents in the workspace are not written, these constituents will be deleted. The single digit zero as the only constituent in the right half will cause everything found by the left half to be deleted. The single digit zero is never entered in the workspace.

New constituents will be inserted in any desired place in the workspace when they are written complete with symbol and any desired subscripts and values in the desired place in the right half.

The computer will add or alter subscripts when they are written on a constituent or number in the right half. If this constituent already has a logical subscript with the same subscript name as the one that is being added, the two subscripts are combined in a special way called dispatcher logic. If there is no overlap in values, that is, if the two subscripts do not have any values in common, the old subscript is replaced by the new one. But if the two subscripts have any values in common, only the values that are common to the two will be retained. An example is shown in figure 9.

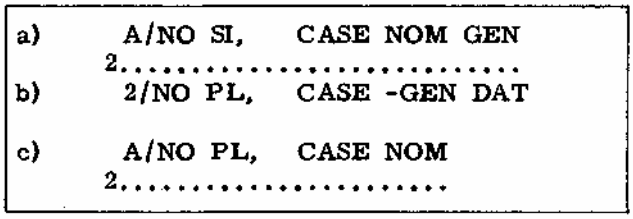


Fig. 9. Example of the combining of subscripts by dispatcher logic. a) shows the number two constituent in the workspace, b) shows the entry in the right half, c) shows the resulting number two constituent in the workspace.

A logical subscript written in the right half with *C in place of its values complements the values of the subscript found in the workspace, that is, all the values that it has are replaced by just those values that it doesn't have. In other words, *C effectively adds a minus sign in front of the subscript values. In the case of numerical subscripts, the new value replaces, increases, or decreases the old depending on whether the value written in the right half follows the period immediately or with an intervening I or D. Since numbers are treated modulo 2^{15}, 1 added to 2^{15} - 1 will give 0, and 1 subtracted from 0 will give 2^{15}- 1. Subscripts will be deleted from a constituent when they are preceded by minus signs in the right half. A dollar sign preceded by a minus sign will cause all subscripts on that constituent to be deleted. Subscripts are added, altered, or deleted in the order from left to right in which they are written in the right half. The same subscript will be altered several times if several expressions involving it are written in the right half.

The computer will carry over subscripts from any single numbered constituent in the workspace to any other single numbered constituent indicated by the right half. For this purpose a subscript name in the right half is followed by an asterisk and a number indicating the number of the constituent from which the subscript is to be carried over. Carried over subscripts go onto the new constituent in the order from left to right in which they are written in the right half. Logical subscripts go onto the new constituent with dispatcher logic. Numerical subscripts carried over either replace, increase, or decrease the old value depending on whether . or .I. or .D. precedes the asterisk. A dollar sign preceding the asterisk will cause all the subscripts from the indicated constituent to be carried over.

After all of the operations indicated by the right half have been carried out on the constituents in the workspace, the numbered constituents remaining in the workspace and any new ones that have been added are given new temporary numbers by the computer in the order in which they are represented in the right half. These new temporary numbers will be of use when the routing is executed.

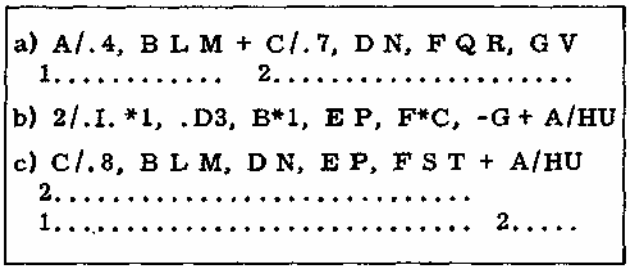


Fig. 10. An example of some right-half operations, a) the numbered constituents in the workspace initially, b) the right half, c) the numbered constituents in the workspace finally, and after renumbering.

An example of some of the operations indicated by a right half is given in figure 10. In this example, the number one constituent in the workspace is deleted. The number two constituent has its numerical subscript increased by the numerical subscript carried over from the number one constituent, and then decreased by 3 to give 8 (7+4-3 = 8). The B subscript is carried over from the number one constituent, the D subscript, not being mentioned, remains unaltered. The E subscript is added from the right half. The F subscript has its values complemented. (We assume that its possible values are Q, R, S, and T.) The G subscript is deleted. Finally, a new constituent is added to the workspace and the constituents in the workspace are renumbered.

The English reading of the right half involves only a few new wordings for abbreviations. These will be found in the section on English reading.

— routing —

The function of the routing section of the rule is to alter the contents of the dispatcher, control input and output functions, direct the computer to search a list, and add or remove plus signs in the workspace.

Dispatcher entries may be written in the routing section. When the routing part of the rule is executed by the computer, these entries are sent to the dispatcher where they combine with the entries there according to dispatcher logic. Logical subscripts on a constituent in the workspace may also be sent to the dispatcher as dispatcher entries. Conversely, dispatcher entries may be carried over as subscripts onto a constituent in the workspace. This latter, to return to the right half for a moment, is done by using the normal notation for carrying over subscripts but by using the letter D to refer to the dispatcher. 1 /CASE*D written in the right half would cause the CASE dispatcher entry to be carried over and added to the number one constituent in the workspace as a subscript. 2/$*D written in the right half would cause all of the dispatcher entries to be carried over as subscripts onto the number two constituent in the workspace. If the constituent in the workspace already has subscripts of the same kind, the dispatcher entries are combined with them according to dispatcher logic.

*D followed by a number in the routing section will cause all of the subscripts on the indicated numbered constituent in the workspace to be sent to the dispatcher as dispatcher entries where they combine with any entries already there according to dispatcher logic. When the computer executes a rule, subscripts designated in the routing section of the rule and dispatcher entries written directly in the routing section of the rule are sent to the dispatcher in the order in which they are written from left to right in the routing section. This is done after the left and the right halves are executed and before the go-to is executed. When subscripts are sent to the dispatcher from the workspace, they are not deleted from the workspace; when they are sent to the workspace from the dispatcher, they are not deleted from the the dispatcher.

COMIT has a special provision for rapid dictionary search. Dictionary entries may be written in a list which will be automatically alphabetized by the computer. This list may be entered from one or more rules called look-up rules. A look-up rule has two special features: *L in the routing section of a look-up rule, followed by one or more numbers referring to consecutively numbered constituents in the workspace, serves to indicate what structure in the workspace is to be looked up in a list. The name of a list, written in the go-to section of the look-up rule, serves to indicate what list the structure is to be looked up in. A list cannot be entered by an automatic transfer of control to the next rule.

When entering a list, the computer temporarily deletes all subscripts from the constituents in the workspace indicated by the *L, and all plus signs between the constituents, thus forming one long symbol. It is this long symbol that is looked up in the list.

The list itself has the following structure: The entries are separate rules. The first rule of a list has a hyphen followed by the name of the list in its name section. The rest of the list rules have nothing in their name sections. List rules have only one subrule each. The long symbol formed by a look-up rule is looked up in the left halves of the list rules. Each left half thus contains only one constituent with a symbol only and no subscripts. Each list rule may also have a right half, routing, and go-to. If the long symbol is found in the list, the corresponding right half is executed in normal fashion. If the number one is written in the right half of the list rule, the long symbol remains in the workspace. If the single number zero is written in the right half, the structure indicated by the look-up rule is deleted. If nothing is written in the right half of the list rule, the items temporarily deleted by the look-up rule are restored and the workspace remains unaltered. If the long symbol is not found in the list, the items temporarily deleted by the look-up rule are restored, leaving the workspace unaltered, and control is automatically transferred to the first rule after the list.

```
A          - + $ + -          //  *L2  B
*                                      C
-B            AND  = . . .             D
           BECAUSE = . . .             E
               BUT = . . .             F
                 .   =  . . .          .
                 .   =  . . .          .
                 .   =  . . .          .
*                                      G
```

Fig. 11. Example of a list rule with look-up rule and two rules to take care of failure to find the indicated structure.

An example of a list is given in figure 11. Rule A is the look-up rule. It serves to find any number of constituents between spaces in the workspace. (Spaces are indicated in the workspace by hyphens.) If the workspace does not have two spaces, the left half is not found and control is transferred to the next rule and then goes to C. If the indicated structure is found, the symbols of the constituents between the spaces are formed into one long symbol which is looked up in list B. If it is not found in the list, control goes to the rule after the list and then to G.

In addition to the look-up rule with its *L abbreviation, there are two other ways of altering the number of plus signs in the workspace.
*K followed by one or more numbers referring to consecutively numbered constituents in the workspace will cause the symbols of these constituents to be compressed into one long symbol, and any subscripts that they may have had will be lost.
*E followed by one or more numbers referring to consecutively numbered constituents in the workspace will cause the symbols of these constituents to be expanded by the addition of plus signs so that each character becomes a separate constituent. A list of characters is given in the center column of figure 12. Any subscripts that the original constituents may have had will be lost.

Only one of the abbreviations *L, *K, or *E may be used in any one rule, and when it is used, it must be last in the routing section to avoid confusion in the numbering of the constituents in the workspace.

The COMIT program communicates with the outside world through input and output functions
under control of abbreviations in the routing section. Reading of input material and writing of output material can be done in any one of several channels and in any one of several formats as follows.

Channels. The particular computer that COMIT is being programmed for (IBM 704) has a number of magnetic tape units connected to it as well as a card reader and punch and a printer. Magnetic tapes may be prepared for the computer from information on punched cards, and material written on tape by the computer may later be read off on a printer or punched on cards. Each input or output abbreviation designates that reading or writing is to take place in channel A, B, C, or one of the others. Then, before the program is run on the computer, the operator connects the channels used by the programmer to various magnetic tape units, printers, etc. Any channel may be connected to any one of several input or output devices. This gives the maximum of flexibility of operation, and allows the output of one COMIT program to become the input of another no matter what channels are designated for input and output in the two programs.

The abbreviations *RW in the routing section followed by a channel designation will rewind the tape unit connected to that channel.

One channel, channel M, is reserved for monitoring purposes and cannot be rewound. It can only be written on. The COMIT programmer can write on this channel any information that may be of use to him later concerning the correct or incorrect operation of his program. Certain information is also written on this channel automatically if the machine discovers certain mistakes in the program during operation.

Material may be read or written in any one of several formats. Format S (specifiers) involves whole constituents, including symbols and subscripts. Format A is for text, and involves only symbols. Both format S and format A are designed for the particular characters available on the printers and card punches in current use. Other formats may be made available if and when other types of input or output equipment become available.

When material is punched on cards for reading into the computer in format S, it is punched in exactly the way that it is to appear in the workspace, including symbols, subscripts, and plus signs between constituents. Any number of characters up to a maximum of 72 may be punched on a card. When material extends over onto another card, the break between cards can be made at any point where a space is allowed, or anywhere in the middle of a symbol.

When the computer executes a rule with an abbreviation in the routing section that calls for reading in format S from a designated channel, the next constituent from the input is brought into the workspace where it replaces the designated numbered constituent. For example, *RSA2 would cause the computer to read in format S the next constituent from channel A and send it to the workspace where it will replace the number two constituent.

When the computer executes a rule with an abbreviation in the routing section that calls for writing in format S, the designated numbered constituents in the workspace are written in the designated channel. They are not deleted from the workspace by this process. For example, *WSM3 5 would cause the computer to write in format S in channel M the number three and the number five constituents from the workspace.

The computer will start a new line or card each time it executes an abbreviation calling for writing in format S. Each line requiring more than 59 characters will end after the next space, fraction bar, or comma, or before the next plus sign, or after 72 characters, whichever comes first. Lines are thus usually ended at a natural break.

Format A is for text, and involves only material written in the symbol sections of constituents . When material is transmitted between the workspace and the input or output channels under the direction of an abbreviation in the routing calling for format A, a special transliteration takes place. The purpose of this transliteration is to allow all of the characters available on the input and output devices to be used in the text. Since many of the available characters have special meanings in the rule — the plus sign separates constituents, the fraction bar separates symbol from subscripts, and so on — these must be represented in a different manner when they are written in the symbol part of a rule if ambiguities are to be eliminated. Accordingly, format A uses the transliteration scheme presented in figure 12.

character on card or tape for format A input	character in workspace, left half, or right half	character on tape or as printed after format A output
letter	letter	letter
.	.	.
,	,	,
space	-	space
number	*number	number
-	*-	-
+	*+	+
=	*=	=
/	*/	/
(	*(	(
)	*)	)
*	**	*
$	*$	$
end of line	*.	end of line
	*letter	letter
	*,	,

Fig. 12. Format A transliteration table. When the text characters of column one are read in by an *RA abbreviation, they appear in the workspace as in column two. When the characters of column two are written out by an *WA abbreviation, they appear in the output as in column three.

Note that the characters available for use in symbols consist of the letters, period, comma,

and hyphen, and an asterisk followed by any character but space.

The first column of figure 12 lists all of the characters available on the printer and card punch. The second column shows how these characters appear in the workspace after they have been brought in by an input operation calling for format A. Note that the letters, period and comma are brought in unchanged, the space becomes a hyphen in the workspace, and all other input characters are prefixed by an asterisk in the workspace. The end of line symbol *. is brought in after the last non-space character on the card.

The second column also lists all possible characters that can be written unambiguously in symbols in a rule. Some of the characters are single and some are double, consisting of an asterisk followed by another character. (An *E expand abbreviation written in the routing does not insert a plus sign between the asterisk and the other character of a double character.)

The third column of figure 12 shows how the characters of the second column will be printed after a write abbreviation calling for format A has been executed. The hyphen is written as a space, *. is interpreted as end of line, or carriage return, all other characters are unchanged except that the asterisk is removed from the double characters. Since the printer can print a maximum of 120 characters in a line, the computer will automatically end a line after 120 characters have been written if the *. abbreviation has not ended it sooner.

When the computer executes a rule with an abbreviation in the routing section that calls for reading in format A from a designated channel, the next character is brought in from the input, transliterated, and entered into the workspace in place of the designated constituent. For example, *RAB2 would cause the computer to read in format A the next character from channel B and send it to the workspace where it will replace the number two constituent.

When the computer executes a rule with an abbreviation in the routing section that calls for writing in format A, the symbols from the designated numbered constituents in the workspace are assembled into a long symbol, transliterated, and written in the designated channel. For example, *WAM1 2 4 would cause the computer to write in format A in channel M the symbols from the number one, two, and four constituents in the workspace. The workspace remains unchanged in this process.

The input and output abbreviations used in the routing section of a rule start with an asterisk followed by R or W for read or write, then there follows a letter designating format A or S, then a letter designating a channel, usually A, B, or C (or M in the case of a write abbreviation only) and finally one number in the case of a read abbreviation and one or more numbers in the case of a write abbreviation designating the numbered constituents in the workspace that are involved. Examples have been given in previous paragraphs.

Summary

This notational system is convenient and well adapted to a large class of problems including language translation and formal algebraic manipulation. The computer automatically converts programs in this notation into actual computer programs. Programs are written in the notation as a series of rules, each of which may have five parts, the name, the left half, the right half, the routing, and the go-to.

An arbitrary rule name may be written in the name section of each rule. In the go-to is written the name of the next rule to be executed.

The material to be operated on exists in the computer as a series of constituents in the workspace. The function of the left half is to indicate which constituents are to be operated on by the computer. This is done by writing in the left half only enough about the constituents or their context to uniquely identify them. In this way, the same rule can be made to apply in a variety of situations that are the same in certain respects. There is a convenient way of locating two or more constituents in the workspace that match each other in a certain way without having to know what the way is in which they match.

If the constituents indicated in the left half cannot be found in the workspace, control goes to the next rule instead of to the rule mentioned in the go-to. This is one type of program branch.

The function of the right half is to indicate what operations are to be performed on the constituents found by the left half. It is possible to add, delete, and rearrange constituents. It is also possible to add subscripts to any constituents, and to rearrange, delete, and calculate with them. There are two kinds of subscripts, numerical subscripts that can be used for counting and simple arithmetic operations, and logical subscripts that can conveniently be used for logical calculations. Both types of

subscripts may be used in the left half to help indicate the material to be operated on. They can thus enter into the condition for a program branch. Logical subscripts can in addition be sent to the dispatcher where, as dispatcher entries, they become effective in controlling n-way program branches. Each dispatcher entry controls which of several subrules is to be carried out in a given rule.

A third type of program branch is provided by the facility for looking up material from the workspace in a list expressed as a series of list rules. This facility can be used for dictionaries. The computer will automatically alphabetize all material in lists to facilitate the look-up operation.

The function of the routing section is to control input and output operations, to control flow of information to and from the dispatcher, to control list look-up operations, and to bring several constituents together into one constituent, or separate a constituent into several constituents, one for each character.

Input and output facilities provide the maximum of convenience for the user. In addition, the system has a number of checks built in that will help the programmer find any mistakes he may make in writing his program.

How to Read a Rule in COMIT

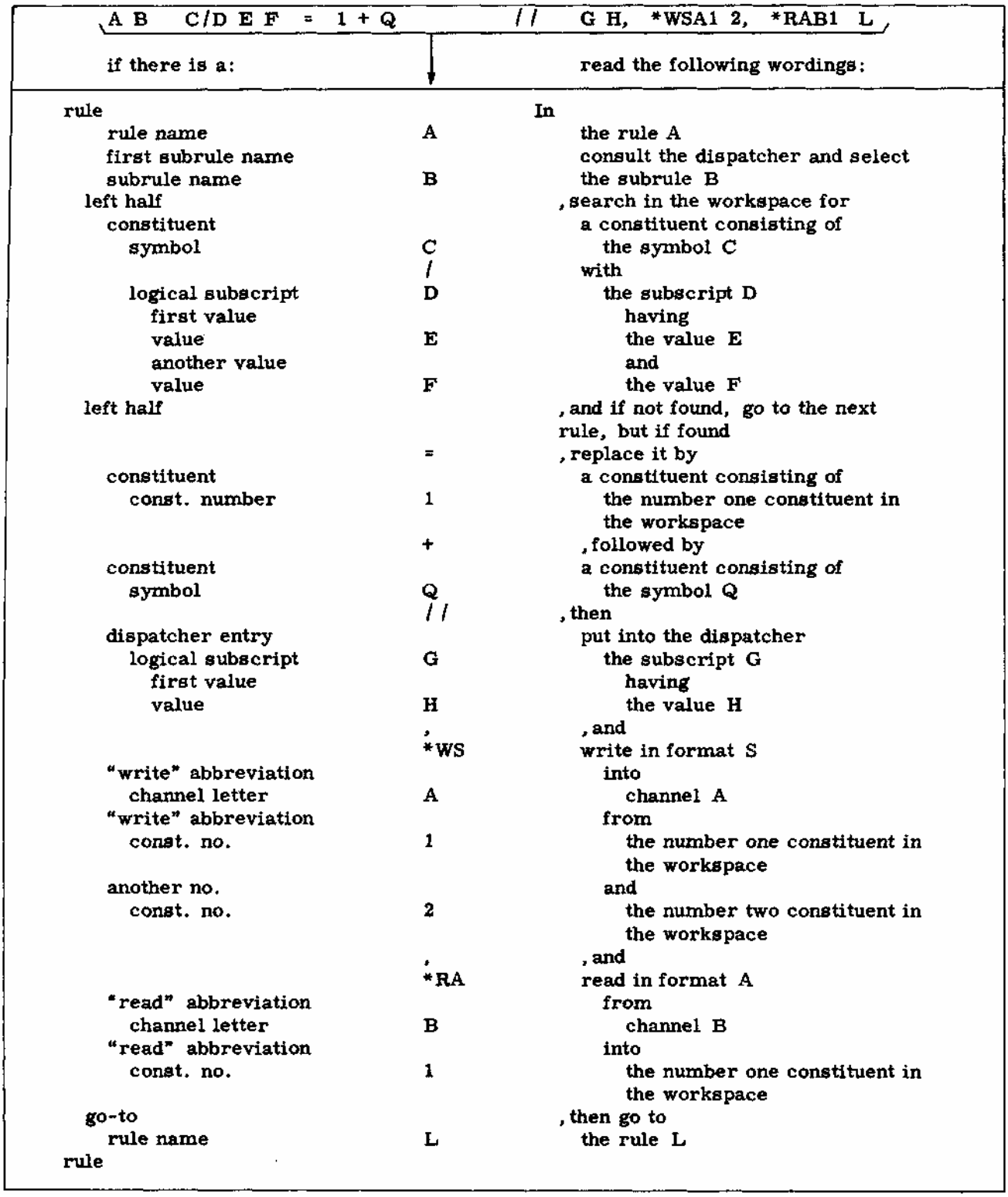
The purpose of this section is to present a summary of the various conventions used for reading a rule of COMIT in English. The readings are, of course, purely mnemonic, for they cannot describe completely what the computer does when it executes the rule.

The various abbreviations used in a rule are tabulated in figure 13. Some abbreviations have several different English readings depending on what part of the rule they are in. When this is the case, a note has been inserted in the table to give an indication of the contexts in which the abbreviation should be given the various readings.

In addition to the English readings associated with the abbreviations, there are conventional wordings that are not associated with any particular abbreviations, but instead with certain positions in the various sections and parts of the rule. In order to summarize these conventional wordings, figure 14 presents a sample rule and its complete reading. The wordings that are associated with the format are provided with an explanatory note giving the circumstances under which they are used.

=	, replace {it / them} by
+	, followed by
/	with
//	, then
-	the list (in column 1) deletion of (before a subscript) (when at the right end of a line, it is a hyphen, rule continues on next line) all values except (before a value)
,	, and (in the routing) , and with (in a constituent)
.	the numerical subscript
*	this rule (in column 1) carried over from (in subscript) the next rule (in go-to)
*C	with all values complemented
*D	put into the dispatcher the subscripts from
*E	expand
*K	compress
*L	look up in a list
*RA_	read in format A from channel _ into
*RS_	read in format S from channel _ into
*WA_	write in format A into channel _ from
*WS_	write in format S into channel _ from
*RW	rewind the tape unit of channel
$	any number of constituents all subscripts
$1	any symbol
$3	three constituents (etc.)
0	nothing
3	three (after., I, D, G, or L) the number three constituent in the workspace (etc.)
D	decreased by
G1	greater than
I	increased by
L	less than

Fig. 13. Abbreviations used in COMIT and their English readings.



A B C/D E F = 1 + Q	//	G H, *WSA1 2, *RAB1 L
if there is a:		read the following wordings:
rule		In
rule name	A	the rule A
first subrule name		consult the dispatcher and select
subrule name	B	the subrule B
left half		,search in the workspace for
constituent		a constituent consisting of
symbol	C	the symbol C
	/	with
logical subscript	D	the subscript D
first value		having
value	E	the value E
another value		and
value	F	the value F
left half		,and if not found, go to the next rule, but if found
	=	,replace it by
constituent		a constituent consisting of
const. number	1	the number one constituent in the workspace
	+	,followed by
constituent		a constituent consisting of
symbol	Q	the symbol Q
	//	,then
dispatcher entry		put into the dispatcher
logical subscript	G	the subscript G
first value		having
value	H	the value H
	,	,and
	*WS	write in format S
"write" abbreviation		into
channel letter	A	channel A
"write" abbreviation		from
const. no.	1	the number one constituent in the workspace
another no.		and
const. no.	2	the number two constituent in the workspace
	,	,and
	*RA	read in format A
"read" abbreviation		from
channel letter	B	channel B
"read" abbreviation		into
const. no.	1	the number one constituent in the workspace
go-to		,then go to
rule name	L	the rule L
rule		

Fig. 14. Conventional wordings that are associated with the format of a rule. The left hand column names the various sections and parts of the sample rule with which the wordings of the last column are associated.

How to Write a Rule in COMIT

The purpose of this section is to present the conventions that must be adhered to when writing a COMIT rule.

General: The left hand 72 columns of the punched card are available for writing COMIT rules. The other 8 columns can be used for numbering the cards if so desired. If a rule requires more than 72 columns to write, a hyphen may be used at the end of one card and the rule continued on the next card in any column. To indicate a space between the hyphenated parts of the rule, leave a space before the hyphen.

Comments enclosed in parentheses are interpreted by the computer as spaces. No parentheses may be included within a comment. A comment continued onto the next card should be hyphenated.

Name section: The first subrule of a rule has a rule name starting in column one. A rule that is never referred to by name in a go-to or in the dispatcher may have an asterisk in column one instead of a name.

All subrules of a rule with more than one subrule have a subrule name. The subrule name is separated from the rule name by one or more spaces, otherwise it starts in any column after the first. A rule can have a maximum of 36 subrules. If there are several rules with the same rule name, they must have identical sets of subrule names.

The first rule of a list has a hyphen in column one followed by the list name. The rest of the rules in a list have nothing in the name section.

A name consists of 12 or fewer consecutive characters. The characters available are the letters of the alphabet, the numbers, and the period and hyphen in medial position, that is not at the beginning or end of the name.

Left half: The first subrule of a rule carries the left half if there is one. All list rules have a left half and only one subrule. The left half is separated from the name by one or more spaces, otherwise it starts in any column after the first.

When the left half could be confused with a subrule name, it should be followed by an equal

Example of subscript	Left half or right half	Binary representation of values or English reading of subscript
SUB	L R	000000...
SUB VAL	L R	100000...
SUB VAL VAL	L R	110000...
SUB -VAL VAL	L R	001111...
SUB -	L R	111111...
-SUB	R	deletion of/the subscript/SUB
SUB*4	R	the subscript/SUB/carried over from/the number four constituent in the workspace
$*4	R	all subscripts/carried over from/the number four constituent in the workspace
-$	R	deletion of/all subscripts
SUB*C	R	the subscript/SUB/with all values complemented
.3	L R	the numerical subscript/three
.G3	L	the numerical subscript/greater than/three
.L3	L	t. n. s./less than/three
.D3	R	t. n. s./decreased by/three
.I3	R	t. n. s./increased by/three
-.	R	deletion of/the numerical subscript
.*4	R	t. n. s./carried over from/...
.I.*4	R	t. n. s./increased by/t. n. s./carried over from/.
.D.*4	R	t. n. s./decreased by/t. n. s./carried over from/.

Fig. 15. A tabulation of all the types of subscripts allowed in the left and the right halves of rules.

sign to resolve the ambiguity. The possible ambiguity is between a left half consisting of a symbol with no subscripts in a rule with no subrule name or right half, and the subrule name of a first subrule with no left or right half.

The left half consists of one or more constituents separated by plus signs and optional spaces. A constituent may be a symbol or $1 with or without subscripts, or it may be a definite or indefinite dollar sign without subscripts, or it may be a number, without subscripts, referring to a numbered constituent already found in the workspace.

The left half of a list rule consists of a single constituent composed of a symbol only.

A symbol is any uninterrupted sequence of characters. A character in a symbol may be a letter; period, comma, or hyphen, or an asterisk followed by any character except space. These latter double characters are treated as single characters by the *E abbreviation. The characters have been summarized in figure 12.

If a constituent has subscripts, these follow the symbol and are separated from it by a fraction bar and optional spaces. Subscripts are separated from each other by commas and optional spaces.

A logical subscript has a subscript name written like a rule name. If it has values, these have the form of subrule names and are separated from it and from each other by one or more spaces. A logical subscript need not refer to a rule name, but if it does, its Values are restricted to the subrule names of that rule.

The types of logical and numerical subscript expressions available for use in the left half are tabulated in figure 15 and indicated by an L. The table also gives an indication of the meaning of the subscripts and how the logical subscript values are stored in the computer in terms of zeros and ones.

Right half: Any rule that has a left half may have right halves in its subrules. Each right half is marked by a preceding equal sign and optional spaces.

The right half consists of one or more constituents separated by plus signs and optional spaces. A constituent in the right half may be a symbol with or without subscripts, or it may be a number, with or without subscripts, referring to a numbered constituent in the workspace. The types of logical and numerical subscripts available for use in the right half are also listed in figure 15, and indicated by an R.

Routing section: The routing section, if written, is preceded by two fraction bars and optional spaces. In the routing section, dispatcher entries may be written in the same way that subscripts and values are written in the right half. In addition the input abbreviations *RAA, *RAB, etc., and *RSA, *RSB, etc. may be written followed by a number designating one numbered constituent in the workspace. The output abbreviations *WAA, *WAB, etc., and *WSA, *WSB, etc. may be followed by one or more numbers referring in any order to numbered constituents in the workspace. The *L, *K, and *E may be written followed by one or more numbers referring to consecutively numbered constituents in the workspace. The numbers are separated by one or more spaces. Separate entries in the routing section are separated by commas and one or more spaces. Only one *L, *K, or *E abbreviation may be written in any rule, and it must be the last thing written in the routing section.

Go-to: In the go-to is written either the name of the rule or list that is to be executed next, or an asterisk signifying that the next rule in sequence is to be executed next. The go-to is separated from the rest of the rule by one or more spaces.

The author wishes to express his appreciation to S. F. Best, F. C. Helwig, G. H. Matthews, A. Siegel, and M. R. Weinstein for their many helpful criticisms and suggestions.

Appendix

Some Sample Programs

We now present a few simple programs written in COMIT. These programs have been chosen for their illustrative and pedagogical value. In order to see how the computer carries out these programs, the reader may have to keep track of the contents of the workspace and dispatcher on a separate piece of paper while going through the programs.

The first seven examples show how some simple operations on text can be carried out. The first one will bring 25 characters of text into the workspace from the input. The remaining six will insert position markers in various places between the characters in the workspace or make various substitutions or order changes. The position markers must be chosen in such a way that they will not be confused with other constituents.

The ninth example is a simple word-for-word translation routine. The text is brought in a character at a time, and each character is looked up in a list to see if it is a letter or mark of punctuation. Each continuous string of letters between punctuation marks or spaces is looked up in the dictionary. The punctuation marks and spaces are carried over into the output text unchanged. Any word that is not found in the dictionary is printed in its original form and enclosed in parentheses. Alternative meanings are separated by fraction bars. An output line is printed as soon as a word is translated that makes the line exceed 55 characters in length. A slight additional complication would be needed to prevent a line from starting with

```
(1.    READ 25 CHARACTERS OF TEXT FROM CHANNEL B)
START          $ = 1 + Q/.0              BRING-IN
BRING-IN Q/.L25 = A + 1/.I1 //*RAB1      BRING-IN
*              Q = 0                              *

(2.    ADD A QD AFTER THE FIRST DOUBLE LETTER)
*         $1 + 1 = 1 + 2 + QD                     *

(3.    ADD A QF AFTER THE FIRST A IN THE WORKSPACE)
*              A = 1 + QF                         *

(4.    ADD A QL AFTER THE LAST A IN THE WORKSPACE)
*              $ = QL + 1                         *
A      QL + $ + A = 2 + 3 + 1                     A

(5.    REPLACE EVERY A IN THE WORKSPACE WITH A B-
       AND EVERY B WITH AN A)
1              A = QR                             1
2              B = A                              2
3             QR = B                              3

(6.    DELETE EVERY E BEFORE A LETTER MARKED WITH A-
       VOWEL SUBSCRIPT)
                (SUGGESTED BY D. DINNEEN)
DEL    E + $1/VOWEL = 2                       DEL

(7.    MOVE THE SECOND CONSTITUENT TO LAST PLACE-
       AND THE ORIGINALLY-LAST CONSTITUENT TO FIRST PLACE)
                (SUGGESTED BY S. ROGOVIN)
*                       $ = 1 + QM                 *
*      $1 + $1 + $ + $1 + QM = 4 + 1 + 3 + 2       *


(8.    PROGRAM TO BRING IN A NUMBER OF CONSTITUENTS AND PRINT OUT-
       ALL OF THE N FACTORIAL PERMUTATIONS OF THEM)
                (PROBLEM SUGGESTED BY B. ULVESTAD)
INPUT                                $ = Q + 1 + N      //*RSA1          INPUT
*                                    N = QL + QR/.1                        *
NUMBER Q + $ + $1 + QL + $ + QR + N = 1 + 2 + 4 + 3 + 5 + 7/.*6 + 6/.I1  NUMBER
*                Q + QL + $ + N + $ + QR = 3 + 2 + 6/.1 + 4 + 5            *
PRINT                          $1 + QL = 2 + 1          //*WAB2           PRINT
*                  QL + $ + $1 + QR + N = 2 + Q + 3 + 1 + 5 + 4            *
PERMUTE $1 + Q + $1 + $ + QL + $ + QR + N = 3 + 2 + 4 + 1 + 5 + 6 + 7 + 8/.D1 *
TEST             QL + $ + QR + $1/.G0 = 1 + 3 + 2 + 4                  NUMBER
MOVE              $1 + Q + $ + QR + $1 = 2 + 1 + 3 + 5 + 4             PERMUTE
```

a space or mark of punctuation, and to allow for the hyphenation of long words at the end of the line.

The eighth example illustrates another class of problems that COMIT is convenient for, that is, problems of an algebraic or manipulational nature.

Readers who would like to use the COMIT system should correspond with the author for further details.

```
(9.    WORD-FOR-WORD TRANSLATION ROUTINE)
PREPARE            $ = 1 + QR                          // WAS PUNC          READ
READ              QR = A + 1                           // *RAA1, *L1   WORD-BOUNDS
DONE                                                                    LAST-LINE
-WORD-BOUNDS       .                                                      IS-PUNC
                   ,                                                      IS-PUNC
                   -                                                      IS-PUNC
                  *. = 0                                                    READ
                  *=                                                     IS-PUNC
                  *-                                                     IS-PUNC
                  *+                                                     IS-PUNC
                  *(                                                     IS-PUNC
                  *)                                                     IS-PUNC
                  *$                                                     IS-PUNC
                  **                                                     IS-PUNC
                  */                                                     IS-PUNC
                  *0                                                     IS-PUNC
                  *1                                                     IS-PUNC
                  *2                                                     IS-PUNC
                   .                                                        .
                   .                                                        .
                   .                                                        .
                  *9                                                     IS-PUNC
OTHERS                                                                 IS-LETTER
IS-PUNC                                                                     *
WAS LETR                                               // WAS PUNC      LOOK-UP
    PUNC                                                                   READ
IS-LETTER                                                                   *
WAS LETR                                                                   READ
    PUNC     $1 + QR = QS + 1 + 2                      // WAS LETR         READ
LOOK-UP      QS + $ + $1 + QR                          // *L2          DICTIONARY
-DICTIONARY    ABER = BUT                                                   A
                 UND = AND                                                  A
               MANN = MAN                                                   A
               FRAU = WOMAN*/WIFE*/MRS.                                     A
                KIND = CHILD                                                A
              KINDER = CHILDREN                                             A
                  .      .                                                  .
                  .      .                                                  .
                  .      .                                                  .
NOT-FOUND    QS + $ + $1 + QR = *( + 2 + *) + 3 + 4                    LINE-PRINT
A            QS + $1          = 2                     // *E1          LINE-PRINT
LINE-PRINT       $56 + $ + QR                         // *WAB1 2         DELETE
TOO-SHORT                                                                  READ
DELETE                $ + QR = 2                                           READ
LAST-LINE             $ + QR = 1                      // *WAB1               *
```

Bibliography

Martin H. Weik 135
A Minimum "Ones" Binary Code for English Text
Ballistic Research Laboratories, Technical Note No. 1215, September 1958

A binary numerical code for alphabetic letters is proposed wherein the number of ones occurring in the binary number varies inversely with the frequency of occurrence of its corresponding letter in English texts. The ratio of holes to total bit locations in punched cards or tape for data storage is then c. 3 to 10. This code is recommended for savings in power, wear, and trouble shooting in punching equipment.

G. H. Matthews

136
T.N. Moloshnaya, V.A. Purto, I.I. Revzin and V.Yu. Rozentsveyg
Certain Linguistic Problems in Machine Translation
Voprosy Yazykoznaniya, no.1, 1957, pp.107-113

This paper is a careful review of the mechanical translation literature together with a discussion of the various requirements and features that a successful mechanical translation program must have.

V. H. Yngve

Erwin Reifler 137
Some New Terminology
Mechanical Translation, vol. 4, no. 3, pp. 52-53

MT research requires cooperation between engineers and linguists. It is important, therefore, to develop a uniform linguistic terminology that can be understood and used by engineers. Furthermore, it is necessary that linguists develop an understanding of the engineering problems involved. The results of cooperation between linguists and engineers working with the MT Pilot Model at the University of Washington are presented here.

Author

B. Zacharov 138
A Refinement in Coding the Russian Cyrillic Alphabet
Mechanical Translation, vol. 4, no. 3, pp. 76-78

By reducing the number of characters to be coded the problem of devising a numerical code for the Cyrillic alphabet can be simplified. This reduction can be achieved by providing code-words for only the lower-case forms of characters that do not occur initially; by disregarding the diacritic of the character ë, and by disregarding the character ë entirely. Ambiguities that arise in the latter cases can be resolved by an examination of the context.

Author

V. H. Yngve 139
A Framework for Syntactic Translation
Mechanical Translation, vol.4, no. 3, pp. 59-65

Adequate mechanical translation can be based only on adequate structural descriptions of the languages involved and on an adequate statement of equivalences. Translation is conceived of as a three-step process: recognition of the structure of the incoming text in terms of a structural specifier; transfer of this specifier into a structural specifier in the other language; and construction to order of the output text specified.

Author

Kenneth E. Harper 140
Contextual Analysis
Mechanical Translation, vol. 4, no. 3, pp. 70-75

Ambiguity, both syntactic and semantic, a problem that arises in the translation of Russian to English because of polysemantic forms in Russian, can be resolved by an analysis of the context in which the polysemantic form occurs. This requires a systematic study of context so that word classes which determine the value of ambiguous forms can be established.

Author

A. Koutsoudas and A. Halpin 141
Research in Machine Translation: II, "Russian Physics Vocabulary, with Frequency Count" (Project MICHIGAN, No. 2144-312-T)

This report contains an alphabetized list of words, with frequencies, occurring in a Russian physics text of 73, 364 running words taken from the Zhurnal Eksperimental'noy i Teoreticheskoy Fiziki. The first volume in which the words are alphabetized from left to right provides a basic breakdown of language data for general research and a compilation of frequencies to be used in examining the problem of dictionary arrangement. The second volume in which the words are alphabetized from right to left provides raw data for a morphological analysis of compound words and derivational and inflectional suffixes.

J. R. Applegate

142
Tezisy Konferencii po Mashinnomu Perevodu
printed by the Ministerstvo Vyshego Obrazovanija SSSR, 1-i Moskovskii gosudarstvennyi-pedagogicheskii institut inostrannyh jazykov, 1958

This is a collection of abstracts of the papers presented at the Conference on Mechanical Translation, which was held in Moscow, May 15-21, 1958. Some seventy papers were presented by most of the Soviet linguists and computer experts who have worked with MT. The problems discussed range from linguistic theory to algorithms for mechanical translation from particular languages.

V. H. Yngve

Ilse Lehiste 143
Order of Subject and Predicate in Scientific Russian
Mechanical Translation, vol.4, no. 3, pp. 66-67

A study by Kenneth E. Harper indicates that word order in Russian scientific writing is sufficiently similar to that of English to permit word-for-word translation from Russian to English. Further study of Russian texts shows that word order in scientific Russian is sufficiently different to require analysis, for translation purposes, based on form and function rather than on word-for-word correspondence.

Author

David L. Johnson and Robert E. Wall 144
Logical Processing and Context Analysis in Mechanical Translation
The Trend, vol. 10, no. 3, July 1958

This paper is a description of the work in the mechanical translation of general technical literature from Russian to English that is being done at the University of Washington. The work, in many respects, has followed the familiar research and development pattern; theoretical development has been followed to a certain point, the method is automatized and tested, then analysis of results yields further advances upon the theory. The article includes a summary of general problems and more detailed sections on the specific problems of dictionary storage, logical processing and the economics of mechanical translation.

J. R. Applegate

145
H. P. Edmundson, D. G. Hays, R. I. Sutton
Studies in Machine Translation— 3: Resume of Machine Codes and Card Formats
The RAND Corporation, Santa Monica, Calif., ASTIA Document Number AD 156048 (Aug. 1958)

This is the third in a series of studies on machine translation issued by The RAND Corporation. In it a detailed description of punched-card formats and codes designed to facilitate the processing of language data in computers is presented. The formats and the special codes described are those in use at The RAND Corporation for research on the machine translation of scientific texts from Russian to English.

J. R. Applegate

D. Yu. Panov 146
Avtomaticheskii Perevod
Second Edition, Moscow, 1958 (71 pages)

The book presents a general description of the work of the mechanical translation groups of the Academy of Sciences, USSR, and the more general problems which arose. Although most of the text treats English-Russian translation, some mention is made of progress in translating from languages of the Far East. There are chapters dealing with the utilization of computers in mechanical translation, the compilation of the automatic dictionary, and programming.

E. S. Klima

I. K. Belskaja 147
Machine Translation of Languages
Research, vol. 10, no. 10, October 1957
pp. 383-389 (Butterworths, London)

The paper gives a general outline of the achievements of the mechanical translation research group of the U.S.S.R. Academy of Sciences. A brief account of earlier experiments in mechanical translation is presented as well as a summary review of some earlier notions about translation in general. In examples of translation into Russian of English, Japanese, and Chinese material, the paper describes current solutions to problems like that of selecting items for the dictionary and coping with relative and contextual meaning.

E. S. Klima

J. P. Cleave 148
A Type of Program for Mechanical Translation
Mechanical Translation, vol. 4, no. 3, pp. 54-58

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.

Author

Erhard Agricola 149
Elektronische Analyse und Synthese in der Sprachwissenschaft
Forschungen und Fortschritte, Band 32,
Heft 3, März 1958, Akademie-Verlag, Berlin

In the course of the paper the following areas are discussed:
1) the way electronic computers designed for mechanical translation operate,
2) the form in which the material must be entered into the computer,
3) the type of linguistic operations which computers can handle,
4) the nature of the refinement, methodological and practical, which may accrue to linguistics through mechanical translation research.

E. S. Klima

Kenneth E. Harper 150
Semantic Ambiguity
Mechanical Translation, vol. 4, no. 3, pp. 68-69

The extent of the problem of multiple meaning in translation is illustrated in this analysis of a sample page of Russian scientific text. The use of an idioglossary represents only a partial solution to the problem.

Author

151
Andreas Koutsoudas and Assya Humecky
Ambiguity of Syntactic Function Resolved by Linear Context
Word, vol. 13, no. 3, Dec. 1957, pp. 403-414

This is a discussion of the problems involved in formulating instructions that will enable an electronic computer to identify the Russian -o/-e/-ee suffixed adverb, short-form adjective or participle in its respective function as either an adverbial modifier or a predicate adjective, and to supply its correct English equivalent. A set of instructions is given which will recognize and translate correctly the 686 occurrences that Occur in the sample text of 31, 000 running words of Russian scientific text taken from Zhurnal Eksperimental'noj i Teoreticeskoj Fiziki, vol. 28, no. 1, pp. 1-128, 1955.

G. H. Matthews

152
D. Yu. Panov, A. Ljapunov, I. S. Mukhin
Avtomatizacija Perevoda S Odnogo Jazyka na Drugoi
Akademija Nauk SSSR, Moskva, 1956

Some of the earlier work done by the Russian group on mechanical translation is presented in this general discussion of machine translation. The differences in translating literary vs. scientific texts are considered. The basic principles of the work done on French-Russian and English-Russian translations are outlined and the problems encountered in this work are defined. Examples of programming for certain words are given (English noun, Russian verb, german noun and one Japanese word). The importance of the linguistic end of the problem is stressed.

E. S. Klima