The possibility of applying an experimental dictionary and a digital computer to a proofreading application was investigated. Because technical abstracts yield a high concentration of proofreading difficulties, a sample of such text was used for study purposes.

The general features of the dictionary, as well as the main algorithms used for dictionary search and text processing, are discussed. Methods for classifying input words and flagging output words are described. Approximately nineteen thousand words of keypunched abstracts were experimentally processed, with results that are discussed. The verification algorithms are evaluated in light of the results obtained, and recommendations for additional improvement and refinement are then presented.

An automatic dictionary and the verification of machine-readable text*

by E. J. Galli and H. Yamada

As part of a broader study dealing with the conversion of documents into machine-readable form, a procedure for machine-assisted proofreading was developed and investigated. The proofreading function was assumed to include correction of spelling errors and standardization of spelling variants. The purpose here is to describe the general procedure, which was based on an experimental dictionary and various classification and processing algorithms, and to comment on the results of the investigation. A complete description of the larger study and its results appears in a final report.[1]

In addition to the special dictionary, called the "English-English Dictionary," several computer programs were developed. The main functions of the English-English dictionary and these associated programs were (1) to verify every text word and symbol against dictionary entries and (2) to produce an output document in which all words were hyphenated if necessary, corrected in certain cases of misspelling, standardized in the case of spelling variants, and flagged for clerical attention if so required.

* This work was jointly supported by the U. S. Air Force Systems Command (under contract AF 30(602)–2860 with Rome Air Development Center, Griffiss Air Force Base, New York) and the International Business Machines Corporation.

The experiment employed an IBM 7094 coupled to a special-purpose lexical data processing system known in brief as the AN/GYA system. Programmed to perform the dictionary search and some of the data processing, the AN/GYA contained the automatic dictionary, which was stored mainly on a random-access photographic memory (the photostore), partly on a magnetic drum, and partly in a magnetic-core memory. Descriptions of the AN/GYA system, photostore, and table-lookup procedures for natural language processing are given elsewhere.[2–4] The 7094 provided overall control, performed input and output editing functions, interpreted subroutine instructions from the AN/GYA system, and performed a number of other miscellaneous tasks. An IBM 1401 computer equipped with several tapes, a card reader/punch, a paper tape reader, and a 120-character chain 1403 printer was used for preparing input to the 7094 and for producing edited or final copy.

Capabilities for the following functions were designed into the English-English dictionary:

- Automatic hyphenation
- Spelling standardization
- Word compounding
- Correction of common spelling errors

With approximately 56,000 entries, the dictionary consisted of English word stems, English endings, whole words, prefix combining forms, suffix combining forms, spelling standardization entries, spelling-error correction entries, and control entries. Entries were derived from several sources, including *Webster's New Collegiate Dictionary,*[5] the *U. S. Government Printing Office Style Manual,*[6] the *Steno-to-English dictionary,*[7] and various text samples.

Dictionary organization and search

The first requirement for an automatic dictionary is a vocabulary large enough that few correct words will be treated as error words due to absence from the dictionary. At the same time, the size of the dictionary must be compatible with existing technology in both speed and economy. Fortunately, the English language has a certain amount of declension, and many words are generated from stem words by affixing a prefix and/or a suffix. Therefore, if constituents such as prefixes, stems, and endings are listed separately with appropriate generative features, less space is required than if the words were listed in full form. It can be readily shown that a constituent organization requires at least an order of magnitude less storage than a full dictionary.

The most general form for a dictionary entry consists of four fields: argument confix, argument data, function data, and function confix. A *confix* is a string of symbols inserted in the input stream after a match is made that serves to modify the next search. Thus, a confix performs a function similar to that of a branch instruction.

A general entry can be denoted by

$$C_1C_2\ldots C_i(D_1)A_1A_2\ldots A_j(D_2)F_1F_2\ldots F_k(D_3)C_1'C_2'\ldots C_m'(D_4)$$

where $C_1C_2\ldots C_i$ is the argument confix, $A_1A_2\ldots A_j$ is the argument data, $F_1F_2\ldots F_k$ is the function data, and $C_1'C_2'\ldots C_m'$ is the function confix. The characters (D_1), (D_2), (D_3), and (D_4) are control codes that serve as field delimiters.

The argument confix is compared with data stored in a confix area, whereas the argument data field is compared with the input string stored in an input area. When a match occurs, the function data field is transferred to an output data area and the function confix to the confix area. The function confix of a matched entry becomes, in effect, a modifier of the next portion of the input string to be searched. Thus the function confix acts to concatenate successive searches by controlling the information that is prefixed to the input data string. There are several permissible variations on the most general form of an entry; for example, either or both confixes may be absent.

Most entries consist of only argument data, function data, and a function confix that specifies the ending class to search next if the match were on a stem rather than a complete word. For a stem, the next search is modified by the confix to examine only that ending set which corresponds to the matched-on stem class; in this case, we use a pair of arguments and confixes for a depth of two. However, this concept can be extended to greater depth; that is, any number of strings can be successively concatenated by the repeated use of confixes.

To arrange dictionary entries in the file, each argument is treated as a left-justified number that concatenates the numerical codes of the argument characters. The entries are sorted numerically and stored in order, tightly packed. The direction of a dictionary search is always from high to low.

The logic of dictionary processing is as follows.[4] Starting from a character marked by an input pointer, the input is compared with dictionary entries to find the entry whose argument matches the longest string of confix and/or input characters. In other words, a search finds the entry which represents the most significant match possible, in a high-to-low ordered sense. Whenever a match is made, the input pointer is updated to the character following the last matched character of the input data stream. The function data is then transferred to an area demarcated by an original-reference pointer and a current-function pointer.

If the matched-on entry contains a function confix, then this field is transferred to a confix area, and the next search begins at the confix area. During comparison, whenever delimiter (D_1) is encountered, the comparison is forced to continue at the position specified by the input pointer. If the matched-on entry does not contain a confix field, then the next search begins at the updated position of the input pointer. A *match-anything* control character may be used in an argument to mask information during com-

Table 1 Stem-ending classes

Class Designation	*Permissible Endings*	*Examples**	*Confix*
Null-S	-, -s, -ed, -ing, etc.	part, represent, report	[s]
Null-S Noun	-, -s, etc.	law, nation, government	[Ns]
Y	-y, -ies, -ied, -ying, etc.	apply, copy, pity	[y]
Y Noun	-y, -ies, etc.	city, ability	[Ny]
Null-ES	-, -es, -ed, -ing, etc.	possess, search	[es]
Null-ES Noun	-, -es, etc.	church, hero	[Nes]
E	-e, -es, -ed, -ing, etc.	base, produce, demonstrate	[e]
Doubling	-, -s, -(C)ed, -(C)ing, etc. where (C) is the repeated final consonant of the stem.	bar, control, defer	[DBG]

* The underlined portion of each example corresponds to the stem.

parisons. Similarly, a *copy-not* control character may be used anywhere in an entry function to allow skipping-without-modification in the output data stream. These control codes are both designated by the symbol μ.

After a significant match is obtained for an input sequence and all pointers are updated, control normally returns to the lookup routine (to effect the next dictionary search). Exceptions occur if the function is flagged for special treatment. In the event that no significant dictionary entry is found for the input string, a match on a breakpoint entry is obtained. Located at points throughout the dictionary, breakpoint entries contain special flags that send the program to appropriate control routines.

Although the *longest-match search algorithm* works on English text rather well,[8] exceptions such as "*metal*anguage," "*dish*armonious," etc., are handled by a special program called the Forced Shorter Match Program.

The dictionary is capable of verifying not only normal English text, but also punctuation and format control symbols embedded in the input text.

Generative features of the dictionary

Most words consist of what can loosely be termed stems and endings, although strict morphological decompositions are not adhered to. In this sense, some stems are word roots while others are compounds of roots, affixes and roots, and so on. The conventional way of categorizing affixes into prefixes and suffixes was also found inadequate, and we adopted our own classification of affixes into endings, prefixes, suffixes, etc.

stems and endings

Stems and their corresponding ending sets were classified into the eight distinct classes summarized in Table 1. It should be noted that this classification is only a first-order approximation. Ultimately, the English language may require many more word classes to generate not all, but just those stem-ending combinations that

are acceptable. Words that possess no derivatives are listed in complete form with a confix designation [-] to allow for possible compounding.

As an example of the stem-ending generative feature, consider the word "part" belonging to the Null-S class. The ending set for this class is capable of generating the following words, among others: part-, part-s, part-ed, part-ing, part-ly, part-less, and part-er. Thus, the stem "part," as well as each of the six derivatives listed above, may be verified by the dictionary with two searches—one for the stem followed by one for the ending. The entire set of ending entries contributes only a negligibly small amount to the dictionary storage requirement. Therefore, as noted previously, a large compaction factor is attained with this type of dictionary structure as compared to one in which each derivative is listed as a separate entry.

The verification of a stem and ending is accomplished in one segment of processing, consisting of two dictionary references. As an example, matching of the word "copy" involves two dictionary entries, $cop(D_2)cop(D_3)[y](D_4)$ and $[y](D_1)y(D_2)y'(\text{-})\tau(D_4)$. The first entry matches the input string up to "cop," reads out "cop," then introduces the confix $[y]$ before the remainder of the input string. Now the dictionary must match with a modified input stream, $[y](D_1)y\ldots$, which the second entry above matches and reads out $y'(\text{-})\tau$. Here, the apostrophe is a conditional hyphen, (-) is for possible compounding, and τ signals the end of a processing segment.

We chose this example deliberately to illustrate a detail of additional complexity that occurs for a relatively small number of dictionary entries. If we represent "copy" in the above manner, then entries for "cope" must be listed in their complete forms. "Cop" can, at the same time, be a word by itself. To handle this case, "cop" and "cops" should also be in the dictionary. However, it is not possible to have two or more entries with identical arguments. Therefore, for "cop," we use $cop@(D_2)cop(D_4)$. In this entry, @ is a word-terminator code used before punctuation marks, space, and format-control symbols.

Words of the doubling class take ending set [DBG]. When a stem is used with an ending "-ed" or "-ing," the last consonant of the stem must be doubled, such as "bar" into "barred" or "barring," etc. This is accomplished with backup instructions (Δ_n^-) which cause the comparisons on the input string to back up by $n + 1$ character positions, where n is a parameter given for each entry.

compounding

In the English language, many words are generated by the compounding of two (or more) words. A list of some 15,000 commonly compounded words appears in the *U. S. Government Printing Office Style Manual* (pp. 77–120), the majority of which are not in *Webster's New Collegiate Dictionary*. In order to verify compound words, either all such words must be included in the dictionary, or some procedure must allow these words to be formed from their constituents. Our dictionary follows the latter approach.

Consider the word "dishwater." If not listed in its full form, the dictionary will match up to "dish," introducing the confix [Nes]. Since the next search is for "[Nes](D_1) *water,*" and since "water" is not an ending, the longest match will be on [Nes](D_1) as an argument and '(-)τ will be read out. Upon detection of τ, the program scans the output and detects (-), a control flag to initiate a new search for the remainder of the word. Then the entries "water wa'ter [s]" and "[s](D_1)@ τ" yield "dish'(-)wa'ter," the (-) code being later deleted in an output editing stage. (For convenience in these and subsequent examples of dictionary entries, we adopt the convention that (D_2) and (D_3) are represented by "space" and (D_4) is omitted.)

The compounding capability allows for the possibility of an error in space omission between consecutive words. Most endings are listed with a terminating code; non-canonical endings are listed without this code but with a function confix [/-] that allows tentative compounding and then invokes corrective measures at a later stage. Words listed in full form are identified by their unique function confix code. The (-?) control flag signifies a conditional compounding. The validity of the resulting compounding is checked at a later point in the program.

prefixes and suffixes

Over 900 prefixes and 600 suffixes are listed in the dictionary. Most are of Greek or Latin origin, such as "anti-," "demi-," "-logy," "-tomy." Prefixes are stored between an initiator flag (Ip) and a terminator flag (Tp), a typical entry being "octa (Ip)oc'ta'(Tp)." This format allows prefixes to be concatenated with other prefixes, words, or suffixes without calling in compounding subroutines. Since suffixes always terminate a word and do not usually combine with other suffixes, not more than one suffix is allowed after a prefix (or prefixes) or a word. Therefore, the confix [/-] is utilized after a suffix to allow for possible missing-space errors, as discussed before.

hyphenation

Hyphenation rules differ slightly from one authority to another; therefore, *Webster's New Collegiate Dictionary* was used as a guide. Every possible hyphenation point in a function was indicated by a conditional hyphen (coded as an apostrophe). These codes were deleted in final processing unless hyphenation was required for right-margin justification, in which case a conditional hyphen at the most appropriate point was converted to a hyphen. Certain exceptions are easily programmed, such as: (1) leave no less than n letters when hyphenating ($n > 1$), (2) hyphenate proper names for narrow column widths only, etc.

Under certain conditions, some endings require that the hyphenation pattern of a stem be modified due to a change in the accentuation point. As an example, the stem for "compute" is listed as "comput com'put [e]." Since the ending "ing" reads out "'ing," the word "computing" is properly hyphenated as "com'put'ing." The same is true of most other derivatives. But the ending "ation" requires that the hyphenation pattern of the last syllable of the stem be altered, so as to obtain "com'pu'ta'tion." Additional gen-

erative entries are included in the ending sets to modify the final portion of the stem readout whenever this is required.

Other occasions where conditional hyphens are used are (1) between two compounded words; (2) after a prefix (sometimes omitted because of variations, such as "aer'o" in "aer'o'dy'nam'ics" or "aer'om'e'ter"); and (3) before a suffix when it is compounded with a word. There are a number of words which change hyphenation patterns when the part of speech and the accent change, e.g., pro'gress (verb) and prog'ress (noun). These are listed in the dictionary without conditional hyphens.

Various hyphenation algorithms have been reported to be between 70 and 98 percent effective.[9] The hyphenation provisions described above, coupled with hyphenation algorithms for words which cannot be verified by the dictionary but which require hyphenation, can be expected to yield hyphenation accuracies far exceeding those attainable solely by algorithms. For example, if 95 percent of the words (in context) are verified and if the hyphenation algorithm utilized is 95 percent effective, then accurate hyphenation can be expected in 99.75 percent of words hyphenated.

numerals

The dictionary includes entries for the processing of numerals and for checking the boundary conditions of numerals for the possibility of missing-space errors between numerals and words. For the processing of Arabic numerals, the confix [#] is introduced upon matching the first digit of a numeral string. If there are punctuation marks such as a period or comma in a string of numerals, the string is subdivided by @ codes and the substrings are processed as separate segments.

If a space is missing before a numeral string, the preceding word lookup would have resulted in a compound flag (-), (/-), or (-?). Subsequently, the first digit is preceded by (/-). At a later stage the Compound Test Program will insert a dummy space ( ), which in turn will be converted into a space in the final program stage. A space error occurring after a numeral is directly treated by the dictionary, by matching on the entry "[#](D_1) ( )" which inserts the dummy space ( ) at the appropriate point.

The dictionary is not able to verify certain symbols involving numerals, such as "6SN7." It is the function of the Symbolism Test Program (to be discussed later) to recognize the existence of symbols in the input text. However, one case requiring special treatment is that exemplified by "1a," "2a," etc. Since "a" is a valid word, a dummy space would normally be inserted before it. An entry "[#](D_1)*a*@ *a*" is included in the dictionary to prevent this.

Provisions are included to verify the occurrence of Roman numerals in text up to 3999 when written in Roman numeral form. Also included are provisions for rejecting improper sequences of Roman numerals, such as CXLXX.

spelling variations

The dictionary has nearly 2000 entries dealing with words that often present difficulties caused by spelling variations. (The section on Orthography in *Webster's New Collegiate Dictionary* was

used as a guide.) Entries were based on the results of a previous analysis of spelling variations occurring in a large sample (6000 documents) of scientific and technical abstracts. The entries standardize spelling variations to the preferred standard, correct non-acceptable variations, and flag certain spellings which are acceptable but are second-choice preferences.

1. A non-preferred spelling is transformed into the preferred form and flagged {N}; e.g., "moveable {N}mov'a'ble"
2. A British spelling with a preferred U. S. form is transferred into the preferred form and flagged {B}; e.g., "colour {B}col'or [s]"
3. An allowable second-choice preference spelling is not altered but is flagged {U}; e.g., "commandoes {U}com'man'does"
4. Equally acceptable variants are listed in both forms without any flags; e.g., "sirup" and "syrup"
5. Correct spellings of rare or archaic words which could be misspellings of more common words are not altered but are flagged {R}; e.g., "calender {R}cal'en'der" compared to "calendar"

Spelling standardizations were also included for certain prefixes, suffixes, and stem-ending combinations, with the appropriate flags.

spelling error correction

The dictionary has been provided with a degree of automatic spelling error correction capability by the inclusion of close to 2000 entries designed to correct misspellings.[10] In each entry,[11] the argument contains the incorrect spelling and the function contains the correct spelling together with a flag {E} which indicates that a dictionary correction has been performed.

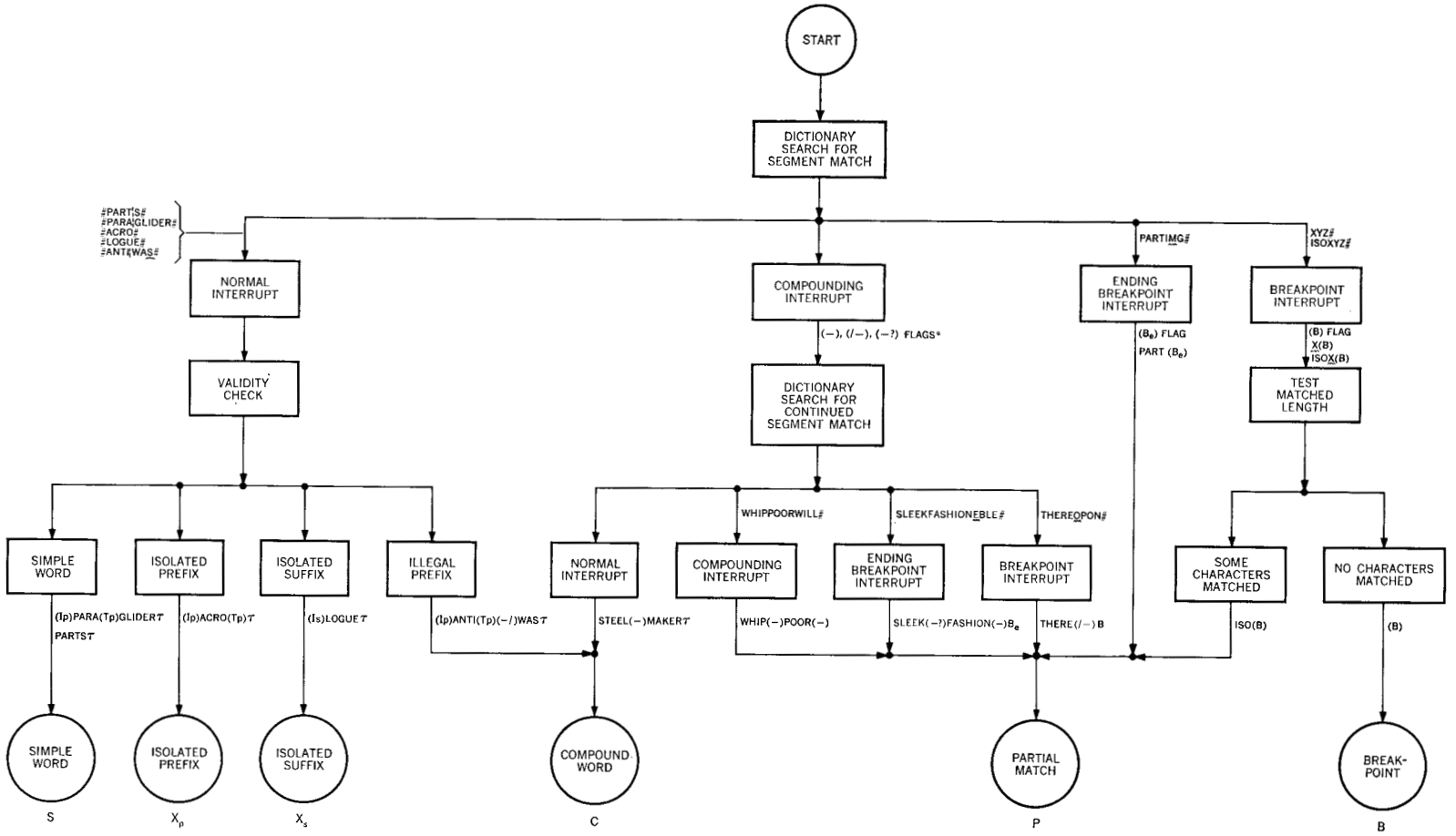
Several other types of entries are included to serve specific functions.

miscellaneous entry types

Breakpoint entries. The function of a breakpoint entry is to notify the control program that an input sequence cannot be verified by the dictionary. There are two classes of breakpoint entries. *Stem breakpoints* have the form "x Δ_0^- (B) τ" for all single-character input codes "x" which have no other significance (punctuation codes are excluded, for example, because they must be appropriately interpreted). (B) is a program control flag indicating that a stem-type breakpoint has been matched. Stem breakpoints for double characters are also included to decrease dictionary search time; e.g., "rk Δ_1^- (B) τ". *Ending breakpoints* are of the form "[e](D_1) (B_e) τ" for confixes [e], [y], [Ny], and [DBD]. The program control flag (B_e) distinguishes this class of breakpoints, indicating that a partial match on a stem has been made but no valid ending can be found.

Capitalized entries. Some 3,500 entries are included to allow the verification of common names of persons, places, etc. They are preceded by a $ code, and are terminated by the word terminator code @ to ensure that a capitalized word will not be partially matched (as, for example, $eliminate and $eli).

Figure 1 Primary word classification program



* FLAGS
(−) COMPOUNDING PERMITTED [e.g. STEEL(−)MAKER]
(/−) COMPOUNDING TO THE RIGHT IS NOT PERMITTED [e.g. THREE(/−)COMPONENT]
(−?) COMPOUNDING VALIDITY MUST BE FURTHER CHECKED [e.g. WHICH(−?)ARE.]
(−?) ALSO PRODUCES THE SEGMENT TERMINATOR T

\# INPUT POINTER USED BEFORE AND AFTER INPUT WORD
¬ ERROR INDICATOR (NOT PART OF THE INPUT)
: WORD SEGMENT DIVIDER (NOT PART OF INPUT)
(−/) COMPOUNDING TO THE LEFT IS NOT PERMITTED

Punctuation and controls. Dictionary entries are also included to properly interpret punctuation and format control symbols that may occur in the input stream, such as period, comma, tabulation, paragraph, etc. These symbols are recognized, transformed into appropriate output codes, and subsequently interpreted by an output editing routine that performs the necessary output composition. Input mnemonics designed to handle Greek and mathematical symbols and specialized formatting (such as centering, underlining, superscripting and subscripting, all capitals, etc.) are treated in a similar manner.

The verification and classification program

This program verifies each input word against the dictionary and classifies it into one of seven categories: simple word, compound word, shorter-match word, proper name, symbolism, foreign phrase, and unclassified word. The unclassified word is a word which does not fall into any of the other six categories, such as a possible error word, or a word missing in the dictionary. There is a program option for either transliterating this class or sending it to the Spelling Error Correction Program (which is not discussed in this paper).

primary program

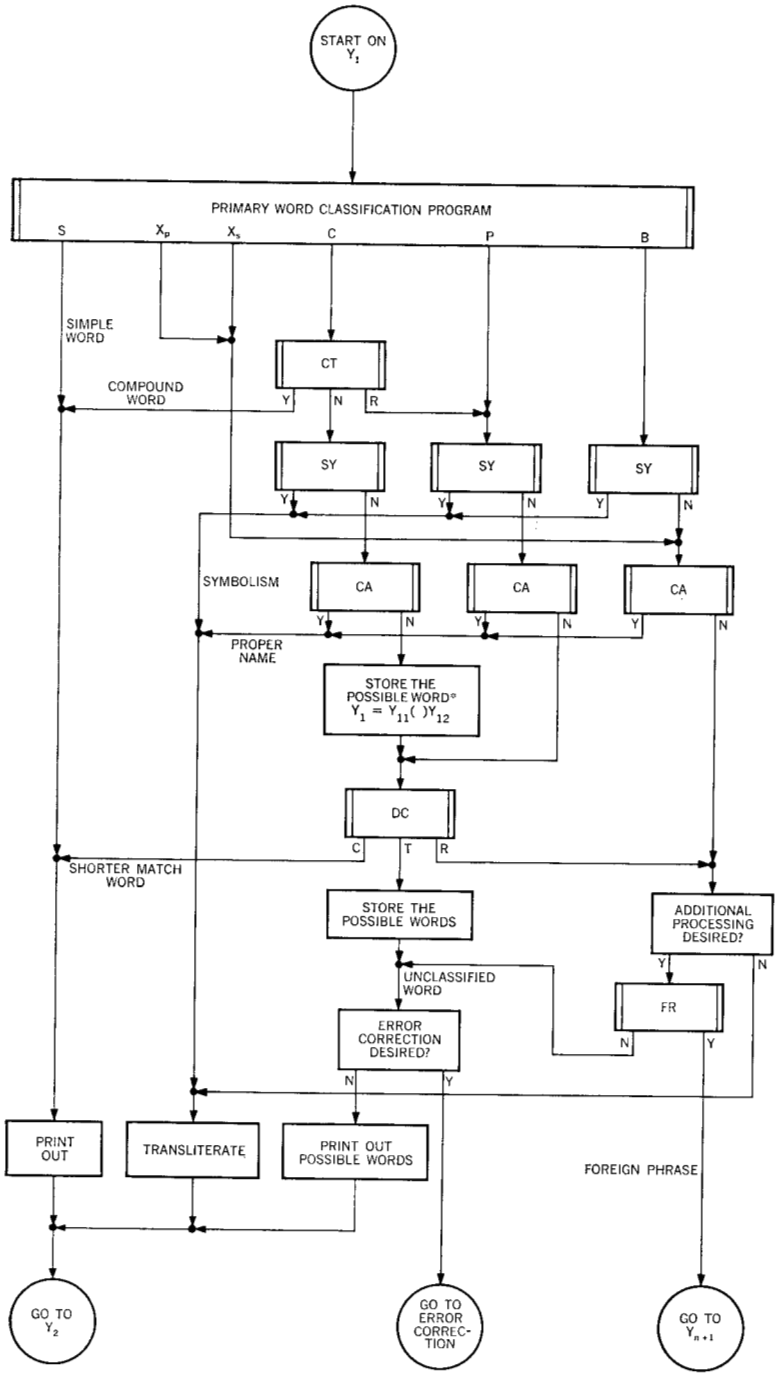
The Word Classification Program is divided into primary and secondary programs. The primary program (Figure 1) proceeds to look up the input stream piece by piece until one of the dictionary entries produces the segment terminator τ. At this point, the dictionary entry defines one of four possible cases: the *normal interrupt* with no flag; the *compounding interrupt* with either one of three flags, (-), (/-), or (-?); the *ending breakpoint interrupt* with flag (B_e); and the *breakpoint interrupt* with flag (B).

Excluding some special cases, a normal interrupt is generally associated with the end of a word. However, some unorthographic cases (improper spellings) may result in a normal interrupt; the program then checks the readout flags and further classifies the result into subcategories. Examples are given in Figure 1. The subclassification serves to expedite further treatment in the secondary program.

A compounding interrupt is generally associated with the end of the first word in a compound word (such as "steelmaker"). The program is so designed that, after detecting this interrupt, it proceeds with the lookup of the rest of the input word. Since this second lookup may terminate in any one of the four interrupt cases, the program is equipped to further classify a word according to the outcome of the second lookup segment. If this results in a normal interrupt, the whole word is classified temporarily as a compound word and is passed on to the secondary program for further processing. For all other cases, the program acknowledges the partial matching and passes the word on to the secondary program.

The ending breakpoint interrupt with flag (B_e) is another case of a partial match which is noted before being passed on to the secondary program. Finally, a breakpoint interrupt with flag (B)

Figure 2 Word classification program



SUBPROGRAMS	DATA	CT OUTPUT	DC OUTPUT
CT — COMPOUND TEST	Y_1 — CURRENT WORD	Y — COMPOUNDING PERMITTED	C — COMPOUND WORD
SY — SYMBOLISM TEST	Y_2 — NEXT WORD	N — COMPOUNDING NOT PERMITTED	T — TWO WORDS
CA — PROPER NAME TEST	Y_{n+1} — (n+1)ST WORD	R — TOTAL REJECTION	R — DECOMPOSITION FAILURE
DC — FORCED SHORTER MATCH			
FR — FOREIGN PHRASE TEST			

* $Y_1 = Y_{11}( )Y_{12}$ CONDITIONAL SPACE SEPARATING THE TWO POSSIBLE SHORTER WORDS Y_{11} AND Y_{12}

signifies either a total failure in matching or a success in matching only up to one or more prefixes. The latter case is treated as a partial match, while the former case is noted separately.[12]

secondary program

The secondary program processes the output of the primary program. Its main subprograms are: Compound Test, Forced Shorter Match, and a set of three routines designed to test for non-English words—the Symbolism Test, the Proper Name Test, and the Foreign Phrase Test. The detailed version (Figure 2) of the Word Classification Program shows the relationship between these subprograms and the primary program.

The function of the Compound Test Program is to test the validity of the words, which the primary program accepted as compound words, by checking their apparent form of juxtaposition.

Without semantic information, such a test is never complete. Nevertheless, we have analyzed the list of commonly compounded words given in the *U. S. Government Printing Office Style Manual* and have arrived at a set of test rules based on certain characteristic properties of compound words, such as (1) not more than two words are compoundable; (2) more than one prefix may be compounded with a word or suffix; (3) certain classes of words should never allow compounding; (4) certain endings do not combine; (5) suffixes of Latin or Greek origin (such as -logue, -cracy, etc.) seldom take another suffix; (6) some suffixes of Anglo-Saxon origin may combine after other suffixes (e.g., -ism-wise); (7) nouns of Anglo-Saxon origin combine with ease (e.g.; arrow-head, etc.)

The Compound Test Program checks for the existence of flags and utilizes rules, such as the aforementioned, to classify the words which are tentatively compounded words into three categories: (1) permissible compounding; (2) compounding not permissible; and (3) total rejection (such as "the-logy").

Most English compound words can be looked up by the longest match technique. However, this algorithm occasionally fails, as exemplified by *bowl*ight, *metal*anguage, etc.[13]

The Forced Shorter Match Program is designed to handle such possibilities. With some exceptions, cases of either total rejection or conditional space separation resulting from the Compound Test Program and cases of partial match resulting from the primary program are subjected to the Forced Shorter Match Program. The basic principle of the program is to insert an asterisk (unique control character) in the input stream which forces a shorter match to occur up to the asterisk if such a match is possible. When such a shorter match is obtained, an attempt is made to match the remainder of the word. By moving this asterisk along the input stream, all possible matches are attempted by the algorithm.

When a lookup of an input word is attempted and fails (i.e., results in a compound test rejection, a partial match, or a breakpoint return), it is suspected first that the particular word may not be an English word. Although we have a set of three crude programs to check such possibilities, an ultimate decision would require semantic information.

The Symbolism Test Program checks the existence of non-word terminating characters consisting of Arabic numerals, certain punctuation marks, etc. The existence of such a character within a word indicates that the word is most likely a code name or symbolism of some kind.

Many capitalized names of common usage are in the dictionary. However, uncommon names which are rejected by the dictionary may be detected by looking for such clues as initial capitalization occurring at some point other than at the beginning of a sentence. Note that this is a necessary, but not a sufficient, test. Our Proper Name Test Program does not check beyond this point, but it is possible to make a number of rather sophisticated tests as Borkowski,[14] and more recently, Altman,[15] have shown.

If a string of several words is rejected by the dictionary, it is highly probable that a phrase of a foreign language is present. The Foreign Phrase Test Program checks for this condition.

The relative positions of the subprograms in the entire system (shown in Figure 2) are a result of a trial and error procedure in our experiment. An optimum structure would be established with reasonable certainty only after analyzing large samples of various kinds of text.

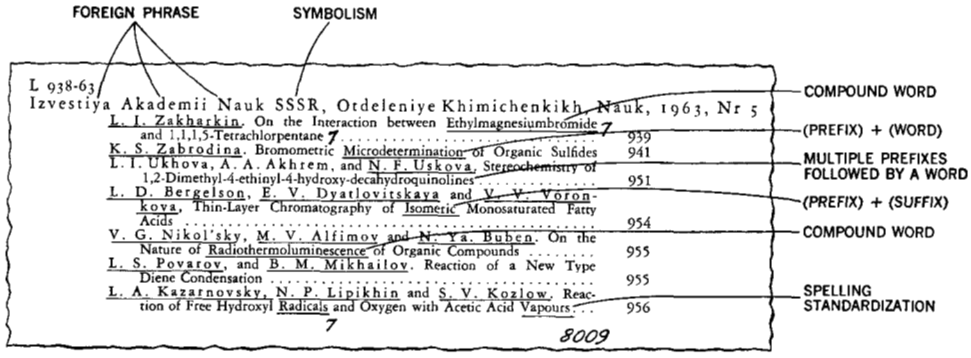
Verification experiment

The experimental dictionary and the Word Classification Program were tested on machine-readable documents from a file of 6000 technical abstracts (furnished by the Air Force). Briefly, the keypunched documents were first subjected to an input pre-edit program in order to make the information compatible with the input requirements of our program. This involved character code conversion, document format changes, end-of-line hyphenation standardization, and other input modifications. Then the documents were subjected to the verification experiment. The results of the dictionary verification were stored on magnetic tape, and later processed with an edit program to obtain copies for inspection.

The main experiment involved 130 documents (about 19,000 words). Of all words processed, 89 percent were properly verified. Of the 11 percent not properly verified, approximately six percent were not found in the dictionary. These were subsequently transliterated, but were also flagged by the Word Classification Program as proper names, parts of foreign phrases (mostly strings of Russian words), or symbols. The remaining five percent were not found in the dictionary, escaped program verification or classification, and were transliterated (and flagged as such). These consisted of: (1) error words or (2) valid words which could not be verified because of inadequacies of the dictionary and/or programs.

Category 1 excludes error words verified and corrected by the dictionary and/or program. The missing-space type of error occurred fairly frequently in the keypunched documents and many of these were corrected. Examples are "keptat," "threecomponent,"

Figure 3 Sample document* with verification examples



* THIS INFORMATION IS FROM A SOURCE DOCUMENT USED IN THE EXPERIMENT AND TYPE SET TO SIMULATE THE DOCUMENT

and "whichare." Some of the more common misspellings were corrected, such as "analyse" and "independant." Many of the spelling variations which occurred were properly standardized, such as "colour," "vapour," and some of the more common less-preferred spellings were detected, such as "grey," "disc," and "inclosure."

Category 2 consisted mainly of foreign-language words that were not capitalized and did not constitute a long-enough phrase to be picked up by the foreign phrase test, specialized English technical terms (such as "betatronic"), and abbreviations such as "cond.," "temp.," etc.

The algorithm for handling compound words processed such rare technical words as "coverbounded," "explosionproof," "hydroxyapatite," "nonmiscible," etc., which are well beyond the range of an ordinary dictionary of only 56,000 words. Some error words, however, may not be verified because of the relatively wide latitude allowed by the compounding capability. An example is "nucieon" (error of "nucleon"). Only by means of a semantic analysis could this sort of error be detected, since "nucieon" is a valid word.

Because of the relatively high occurrence of proper names, foreign words, symbols, and specialized technical terms in these documents, most words of the 11 percent that were not found by the dictionary lookup procedures were non-dictionary words, not input errors. It should be noted that the sample text used in the experiment was quite complex both in nature and in structure. With more standard types of text, one would expect this percentage to be lower. A sample document is illustrated in Figure 3, in which some typical examples of verification types are marked and defined.

Because this demonstration was limited in scope, no quantitative conclusions regarding the potential cost reductions possible with computer-assisted editing were drawn. However, in view of the inaccuracies and costs associated with clerical editing,[16] it seems

likely that further attention will be given to the area.[17] In that event, the approach herein described, the types of problems indicated by the test demonstration, and the generally encouraging results may serve as a guide for other studies.

ACKNOWLEDGMENT

The authors gratefully acknowledge the efforts of F. L. Barone, W. J. Eisner, and K. F. Scharfenberg for providing programming support for the project and for contributing many valuable suggestions.

CITED REFERENCES AND FOOTNOTES

1. *FTD Semi-Automatic File Conversion System*, Final Report on Contract AF 30(602)–2860, Volumes 1, 2, and 3 (March 1965). Available as DDC Documents, AD 476682 (Volume 1), AD 476699 (Volume 2), and AD 476683 (Volume 3).
2. *Computer Storage Integration Group* AN/GYA–( ), Final Report on Contract AF 30(602)–2754 (October 1964). Available as DDC Document AD 451972.
3. G. Shiner, "The USAF automatic language translator, MARK I," *1958 IRE National Convention Record*, Part 4, 296–304.
4. J. L. Craft, E. H. Goldman, and W. B. Strohm, "A table lookup machine for processing of natural languages," *IBM Journal of Research and Development* **5**, No. 3, 192–203 (July 1961).
5. *Webster's New Collegiate Dictionary*, (Based on *Webster's New International Dictionary*, Second Edition), G. & C. Merriam Company (1961).
6. *U. S. Government Printing Office Style Manual*, Revised Edition (1959).
7. E. J. Galli, "The stenowriter—a system for the lexical processing of stenotypy," *IRE Transactions on Electronic Computers* **EC-11**, No. 2, 187–199 (April 1962).
8. G. W. King, "Table lookup procedures in language processing," *IBM Journal of Research and Development* **5**, No. 2, 86–92 (April 1961).
9. See, for example: (a) E. Yasaki, "The computer and newsprint," *Datamation* **9**, No. 3, 27–32 (March 1963); (b) F. J. Damerau, "Automatic hyphenation scheme," U. S. Patent Office, Serial No. 375714 (June 1964).
10. Many of the entries were based on actual document file spelling errors. Many others were collected from errors made by secretaries in typing. The Printing Office Style Manual (Reference 6) contributed some, and the following books were consulted: J. Lasky, *Proofreading and Copy-Preparation*, Mentor Press, (1954); J. O. Bailey, *Proper Words in Proper Places*, American Book Company (1952); N. Foerster, et al., *Writing and Thinking*, Fifth Edition, Houghton Mifflin Company (1952).
11. Many of these entries are for words containing frequently misspelled letter patterns. Some examples are: "preceeding {E}pre'ced'ing," "sacreligious {E}sac'ri'le'gious," "surpriz {E}sur'pris [e]." Words ending in "-ible," "-able," "-eable," and derivatives of these are especially troublesome. Words which take "-ible" and its derivatives are rather few in number. Therefore, all such words are listed in the dictionary in their full correct form and also in misspelled form with the correct function readout, e.g., "abhorrible" and "abhorrable (error)." This ensures that no matter how these words are spelled their correct form is always produced by the dictionary. Error correction ending entries are included to correct errors involving the "-able" ending and its derivatives, e.g., "[es] (D_1)eable@ 'a'ble{E} τ," "[es] (D_1)ible@ 'a'ble{E} τ;" etc. Errors involving the "-eable" class are handled in a similar manner. Ending-error-correction entries are

also included to correct common misspellings of certain endings in each stem-ending class. Some examples are: "[e](D$_1$)efull@ e'ful{E}$_7$" (e.g., wastefull→wasteful). "[e](D$_1$)eing@ 'ing{E}$_7$" (e.g., computeing→computing), etc.

12. The majority of the partial matches are caused (1) by an error in the input word, (2) by the absence of the word in the dictionary, or (3) by the failure of the longest match. The following examples are illustrative.
    (a) missing word: "famulary"→fam-(B$_e$)
    "biopsy"→(Ip)bio(Tp)-(B)
    (b) error word: "fast endotoxin"→"fastendotoxin"
    →"fast-en(/-)-dot-(B$_e$)"
    (c) longest match failure: "wartime"→"wart(-)-(B)"
13. These words, if not in the dictionary in their full form (and they are not in *Webster's Collegiate*), will result in "-ight" or "-anguage" remaining as residues of the first look-up. Because of the flexibility of the language, and because of storage limitations, we cannot include all such possible cases in the dictionary. Besides, failure of the longest match may be caused by errors in the input as well, as exemplified by "addition algorithm" → "additionalgorithm" → "additional-go-rithm" ("rithm" is a rare variant of "rythm").
14. C. G. Borkowski, "A system for automatic recognition of names of persons in newspaper texts," IBM Research Report RC 1563, Research Division, Yorktown Heights, New York (1966). Available as DDC Document AD 481926.
15. E. B. Altman, "On the recognition of personal names in natural text," IBM Research Report RC1871, Research Division, Yorktown Heights, New York (1967).
16. As a particular case in point, in the document conversion study (of which this work was a part) it was found that for the keypunch method of conversion, the cost of manual proofreading and correction constituted 43% of the total conversion cost; and that even though 77% of the transcription errors were detected, only 59% of the errors were properly corrected. For 18% of the errors, corrections were improperly made, and 23% of the original errors were not detected in proofreading. The overall effect of manual proofing was to increase the conversion accuracy from 96.7% to 98.7%, on a word count basis, while the additional effort involved increased the cost by approximately 74 per cent.
17. For a survey of other work that has been done in the field of copyediting, and some indication of the directions for further experimentation and development, see W. A. Danielson, "The Man-Machine Combination for Computer-Assisted Copy Editing," in *Advances in Computers* **VII**, edited by F. L. Alt and M. Rubinoff, Academic Press, New York, 181–193 (1966).