[*Mechanical Translation and Computational Linguistics*, vol.11, nos.1 and 2, March and June 1968]

Development of a Stemming Algorithm*

by Julie Beth Lovins,† Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

A *stemming algorithm, a procedure to reduce all words with the same stem to a common form, is useful in many areas of computational linguistics and information-retrieval work. While the form of the algorithm varies with its application, certain linguistic problems are common to any stemming procedure. As a basis for evaluation of previous attempts to deal with these problems, this paper first discusses the theoretical and practical attributes of stemming algorithms. Then a new version of a context-sensitive, longest-match stemming algorithm for English is proposed; though developed for use in a library information transfer system, it is of general application. A major linguistic problem in stemming, variation in spelling of stems, is discussed in some detail and several feasible programmed solutions are outlined, along with sample results of one of these methods.*

I. Introduction

A stemming algorithm is a computational procedure which reduces all words with the same root (or, if prefixes are left untouched, the same *stem)* to a common form, usually by stripping each word of its derivational and inflectional suffixes. Researchers in many areas of computational linguistics and information retrieval find this a desirable step, but for varying reasons. In automated morphological analysis, the root of a word may be of less immediate interest than its suffixes, which can be used as clues to grammatical structure. (See, e.g., Earl [2, 3] and Resnikoff and Dolby [6]. This field has also been reported on by S. Silver and M. Lott, Machine Translation Project, University of California, Berkeley [personal communication].) At the other extreme, *what* suffixes are found may be subsidiary to the problem of removing them consistently enough to obtain sets of exactly matching stems. Word-frequency counts using stems, for stylistic (as described by S. Y. Sedelow [personal communication]) or mathematical analysis of a body of language, often require matched stems. (So does stemming as part of an information-retrieval system, the specific application which motivated this paper.) But certain linguistic problems are common to any "stem-oriented" stemming algorithm, no matter what its ultimate use. The brief description below of the framework within which Project Intrex is planning to use
a stemming algorithm should be viewed as but one possible application for research on the morphological structure of English and other languages. Similarly, a variety of applications are considered in evaluating the theoretical and practical attributes of several previous algorithms.

As a major part of its information transfer experiments, Project Intrex [5] is developing an integrated retrieval system in which a library user, through a remote computer terminal, can first obtain extensive information from a central digital store about documents that are available on a specific subject, and then obtain the full text of the documents. A prototype retrieval system is being assembled in order to permit experimentation with its various components. The experimental system will use a specially compiled augmented library catalogue containing information on approximately 10,000 documents in the field of materials science and engineering, including not only author, title, and other basic data about each document but also an abstract, bibliography, and a list of subject terms indicating the content of the document. Each subject term is a phrase of one or more English words. A stemming algorithm will be used to maximize the usefulness of the subject terms. In many cases, the information which is semantically significant to the user of the system is contained in the stems of the lexical words in the subject terms, and suffixes and function words merely enable this information to be expressed in a grammatical form. The form of the words which the user inputs will often not correspond to that of the original words in the catalogue. To permit the words in the user's query to match the words in the catalogue entry's subject terms, both query and subject terms can be stripped of the suffixes that prevent their matching. For example, *computational* and *computing* might both be stemmed to *comput.*

In constructing the software needed for this particular application of stemming (or any other), we encounter questions which are answerable only in terms of the over-all system. For instance, what should constitute a "word" to be stemmed? In the case of Intrex, what suffixes should the algorithm search for that are specifically

* The research reported on in this paper was carried out at Project Intrex, which is supported under a grant from the Carnegie Corporation; under contract NSF-C472 from the National Science Foundation and the Advanced Research Projects Agency of the Department of Defense; and under a grant from the Council on Library Resources, Inc.

† Now at the University of Chicago, Department of Linguistics.

oriented toward terms in materials science and engineering? These are questions of less general interest than the linguistic problems of extracting a stem from any one word in a non-specialized vocabulary (for an example of lists of affixes taken from terms in specific technical fields, see Dyson [1]). The development of an efficient algorithm should logically precede investigation of these questions, and they will not be discussed further here.

The approach to stemming taken here involves a two-phase stemming system. The first phase, the stemming algorithm proper, retrieves the stem of a word by removing its longest possible ending which matches one on a list stored in the computer. The second phase handles "spelling exceptions," mostly instances in which the "same" stem varies slightly in spelling according to what suffixes originally followed it. For example, *absorption* will be output from phase one as *absorpt, absorbing* as *absorb.* The problem of the spelling exceptions, which in the above example involves matching *absorpt* and *absorb,* is discussed thoroughly in Section V of this paper. One particular solution to the problem, termed *recoding,* has been implemented in the present phase two. We also plan to use the present basic algorithm as a foundation in testing out other feasible solutions.[1] This plan is appropriate because spelling-exception rules can, and probably should, be formulated independently of the stemming algorithm proper.

II. Stemming, Form, and Meaning

By its computational nature, a stemming algorithm has inherent limitations. The routine handles individual words: it has no access to information about their grammatical and semantic relations with one another. In fact, it is based on the assumption of close agreement of meaning between words with the same root. This assumption, while workable in most cases, in English represents an approximation at best. It is a better or worse approximation depending on the intended use of the stems, the semantic vagaries of individual roots, and the strength of the algorithm (how radically it transforms words). A stemming algorithm strong enough to group together *all* words with the same root may be unsuitable for, say, word-frequency counting. For such applications one would not wish a pair like *neutron-.neutralizer* to coincide, and one would prefer to work with a very limited list of suffixes.

Where stems are used as a means of associating related items of information, as they are in an automated library catalogue, and where the catalogue can be interrogated in an on-line mode, it seems best to use a strong algorithm, that is, one that will combine more words into the same group rather than fewer, thus providing more document references rather than fewer.

[1] I am indebted to Richard S. Marcus and Peter Kugel for valuable discussion of this specific problem and of this report as a whole.

After a word in the library user's query has been stemmed and a matching stem and associated list of full-word forms has been found in the catalogue and presented to the user, he may decide to discard some of these forms in order to inhibit searching for those full-word forms which are unrelated to his subject.

Occasionally, the output of a stemming routine may be not only ambiguous but also "not English." This happens when a suffix is identical to the end of some root. For instance, *-ate* is a noun suffix in *directorate,* but simply part of a verbal root in *create* and *appreciate.* In English, situations of this type limit the use of suffixes as clues to parts of speech. Sometimes grammatical information is required for stemming, not provided by it.

However, the generation of such non-linguistic stems as *cre-* and *appreci-* is not a serious problem; if the purpose of stemming is only to allow related words to match, then the stems yielded by a stemming algorithm need not coincide with those found by a linguist. The exact form of the stem is not critical if it is the same no matter what suffixes have been removed following it, and if "mistaken" stemming does not generate an ambiguity. Similarly, the ending that must be removed in order to achieve a consistent algorithm is determined in relation to the stemming system as a whole. The ending may or may not be exactly equivalent to some entity in English morphology, and it may be acceptable to have the computer program remove it when a linguist would not, with no detriment to the ultimate results.

III. Types of Stemming Algorithms

Two main principles are used in the construction of a stemming algorithm: *iteration* and *longest-match.* An algorithm based solely on one of these methods often has drawbacks which can be offset by employing some combination of the two principles.

Iteration is usually based on the fact that suffixes are attached to stems in a "certain order, that is, there exist *order-classes* of suffixes (see, e.g., Lejnieks [4]). Each order-class may or may not be represented in any given word. The last order-class—the class that occurs at the very end of a word—contains inflectional suffixes such as *-s, -es, and-ed.* Previous order-classes are derivational. (As pointed out by J. L. Dolby [personal communication], there are several cases known in which a derivational suffix *(-ness)* follows an inflectional one *(-ed* or *-ing*). This occurs with certain nominalized adjectives derived from verbs by use of one of these two inflectional endings, for example, *relatedness, disinterestedness, willingness.)* An example of the lowest order-class in a word may be what is technically part of the root (see the *-ate* example above), but for the purposes of computation it is considered part of the ending. An iterative stemming algorithm is simply a recursive procedure, as its name implies, which removes strings in each order-class one at a time, starting at the end of a word and working toward its beginning. No more than one match is allowed within a single order-class, by

definition. One must decide how many order-classes there should be, which endings should occur in each, and whether or not the members of each class should be internally ordered for scanning.

The *longest-match* principle states that within any given class of endings, if more than one ending provides a match, the one which is longest should be removed. This principle is implemented by scanning the endings in any class in order of decreasing length. For example, if *-ion* is removed when there is also a match on *-ation,* provision would have to be made to remove *-at,* that is, for another order-class. To avoid this extra order-class, *-ation* should precede *-ion* on the list.

An algorithm based strictly on the longest-match principle uses only one order-class. All possible combinations of affixes are compiled and then ordered on length. If a match is not found on longer endings, shorter ones are scanned. The obvious disadvantage to this method is that it requires generating all possible combinations of affixes. A second disadvantage is the amount of storage space the endings require.

The first disadvantage may also be present to a large degree when one is setting up an iterative algorithm with as many order-classes as possible. To set up the order-classes, one must examine a great many endings. Furthermore, it is not always obvious to which class a given string should belong for maximum efficiency. It is also entirely possible that the occurrence of members of some classes is context dependent (see below). In short, while an iterative algorithm requires a shorter *list* of endings, it introduces a number of complications into the preparation of the list and programming of the routine.

Some idea of the breadth of these complications is gained through consideration of another basic attribute of a stemming algorithm: it is *context free* or *context sensitive.* Since "context" is used here to mean any attribute of the remaining stem, "context free" implies no qualitative *or* quantitative restrictions on the removal of endings. In a context-free algorithm, the first ending in any class which achieves a match is accepted. But there should presumably be at least some quantitative restriction, in the sense that the remaining stem must not be of length zero. An example of this extreme case is the matching of *-ability* to *ability* as well as to *computability.* In fact, any useful stem usually consists of at least two letters, and often three or four constitute a necessary minimum. The restriction on stem length varies with the ending; how it varies can again only be determined in relation to the total system. The algorithm developed by Professor John W. Tukey of Princeton University (personal communication) associates a lower limit with each ending. Some of his limits are quite high (e.g., seven letters). I have been less conservative and have proposed a minimum stem length of two; certain endings have an additional restriction in that their minimum stem length is three, four, or five letters.

The kind of qualitative contextual restrictions that should be imposed is a somewhat open question. In order to get the best results, certain endings should not be removed in the presence of certain letters in the resultant stem, usually those letters that immediately precede the ending. The more desirable form of context-sensitive rule is a general one that can be applied to a number of endings, but such rules are few. One example is "do not remove an ending that begins with *-en-,* following *-e."* Violation of this rule would change *seen* to *se-,* a potentially ambiguous stem (cf. *sea* minus *-a*, *seize* minus *-ize*, etc.). But a number of rules must be created for individual endings in order to avoid certain special cases peculiar to those endings. One can go to great lengths in this direction, with increasingly small returns. I have preferred to start by treating a number of the more obvious exceptions in the hope that the percentage of words not accounted for will be small enough to preclude the need to add many additional rules.

An iterative stemming algorithm, that is, one that contains more than one order-class of endings, is presumably no less complicated by context-sensitive rules than a one-class algorithm, and is probably more so; exceptions associated with the members of each class may depend on a rather complicated context. For example, suppose there is a rule (in a non-iterative algorithm) stating that minimal stem length is five before *-ionate.* The endings *-ion* and *-ate* occur separately, also, with different restrictions. In an iterative routine, *-ion* and *-ate* would *only* occur as separate endings, in different order-classes; and *-ion* would be restricted by the rule that its preceding context must be of length five *if -ate* was found during the preceding iteration. In other words, the endings that are removed may influence the lower-order endings that can be removed subsequently. The implications for simplicity in programming are self-evident. In a pure longest-match algorithm, the only context that need be considered is the prospective stem itself.

Since computer-storage space for endings was not an immediate problem, it was decided to test a non-iterative stemming algorithm based on a one-class list of endings. That is, the intuitively inefficient procedure of listing both singular and plural forms, and so on, has been followed in order to minimize the number of context-sensitive rules necessary. Compilation of the actual list of endings used is discussed in the next section; the algorithm is outlined in Section VI.

The author is aware of three previous major attempts to construct stemming algorithms. Tukey has proposed a context-sensitive, partially iterative stemming algorithm whose endings are divided into four order-classes. The first (highest-order) class contains only terminal *s* which, however, is not removed after *i, s,* or *u.* The second class is recursive, the third is non-recursive and ordered on length. The fourth class consists of remaining terminal consonants. The last three classes also have a few members each with simple context restrictions, and all classes have limits on minimum stem length. (The basic structure of this "tail-cropping" algorithm

is not affected by its multilingual orientation, though the endings used would obviously differ from those found in a procedure for English only.)

One of the more interesting things about the Tukey system is its structural complexity. One class uses the longest-match principle only, while another is iterative (and thus not a proper order-class). Presumably the object of this heterogeneous structure is to avoid the repetitiveness of a one-class ending list in the most concise way possible. However, as stated earlier, there is a compromise between conciseness of rules and simplicity of programming.

By contrast, the algorithm developed at Harvard University by Michael Lesk, under the direction of Professor Gerard Salton [10], is based on an iterated search for a longest-match ending. After no more matches can be found, terminal *i, a,* and *e* are removed, and then possibly terminal consonants. There are apparently no contextual restrictions of any kind. (A brief description of the algorithm, including a useful list of 194 endings, was transmitted to us via personal communication. A sample of these suffixes, and further information about the algorithm, have more recently appeared in Salton [9].)

A third algorithm has been developed by James L. Dolby of R and D Consultants, Los Altos, California (personal communication). This algorithm works in three stages, the first of which involves a set of context-dependent transformations. Most of the cropping is done in the second stage, a context-free, longest-match, recursive procedure which removes endings in any order but is subject to the restriction of a two-syllable minimum stem length. In the final stage there is a context-dependent dropping of inflectional forms. The endings used were derived by algorithm from word lists on the basis of orthographic context, and are "minimal" segments of one to four letters in length.

IV. Compilation of a List of Endings

A one-class list of endings (concatenations of suffixes) was compiled in the following way: A preliminary list was based on endings found in a small portion of the augmented catalogue being developed by Project Intrex and on endings in the list used at Harvard. The preliminary list was evaluated by applying the endings on this list to a portion of the output from Tukey's tail-cropping routine, levels 1-3, and volumes 5-7 of the *Normal and Reverse English Word List* [8] (volumes 5-7 contain unbroken words sorted alphabetically when written from right to left). Since each of these lists is organized according to ends of words, it was possible to see whether the removal of a given ending would result in (1) two different stems matching, or (2) a stem not matching another stem which it *should* match. Either of these conditions, unless it was caused by a spelling exception or caused improper matching in only a few rare cases, necessitated the addition of new endings, the disposing of old ones, or the addition of context-sensitive rules, until the system seemed adequately self-consistent. The resultant experimental list contained about 260 endings, divided into eleven subsets; the subsets are ordered in disk storage in accordance with decreasing length of the endings and are internally alphabetized for easy handling. The internal order does not affect the end result of the algorithm. Each subset is preceded by a special heading giving the length of the endings in it; each ending is followed by a condition code and a carriage return as delimiter. The condition code consists of a letter of the alphabet containing information about contextual restrictions on the stem preceding the ending.

The present list of endings, which is a slightly modified version of the original one (see Section VI), is given in Appendix A; the context-sensitive tests associated with the endings are listed in Appendix B.

V. Some Cures for "Spelling Exceptions"

The term "spelling exceptions" is a catchall term covering all cases in which a stem may be spelled in more than one way. The majority of such variations in English occur in Latinate derivations. The examples given below show some of the range and type of variations that may occur. Trouble spots are italicized; the stem is separated from the ending by a vertical bar.

produc|er : produc*t*|ion	inver*t*|ed : inver*s*|ion
induc|ed : induc*t*|ion	adhe*r*|e : adhe*s*|ion
induct|ed : induct|ion	regist*er*|ing : regist*r*|ation
consum|ed : consum*pt*|ion	resol*v*|ed : resol*ut*|ion
absor*b*|ing : absor*pt*|ion	admi*tt*|ed : admi*ss*|ion
atten*d*|ing : atten*t*|ion	circ*l*|e : circ*ul*|ar
expan*d*|ing : expan*s*|ion	matrix| : matric|es
respon*d*| : respon*s*|ive	lattic|e : lattic|es
exclu*d*|e : exclu*s*|ion	inde*x*| : indic|es
colli*d*|ing : colli*s*|ion	hypothe*s*|ized : hypothe*t*|ical
	analy*s*|is : analy*t*|ic

Several other types of spelling exceptions also occur, such as the doubling of certain consonants before a suffix (input:input*t*ing), and contrasting British and American spellings (analy*s*ed:analy*z*ed).

While the derivational spelling changes do occur only before certain endings, this set of endings is usually quite large. Thus it is not practical to consider the exceptional stem-terminal consonants as part of the endings in a one-class algorithm such as the one we are using; the number of extra endings that must be included to do so is prohibitive. Two major types of post-stemming procedures may be followed to take care of the exceptions, however. I shall call them *recoding* and *partial matching.* (Salton [9, p. 82] describes a routine which includes some attributes of each of the procedures discussed below. While it will take care of such problems as consonant doubling, it does not appear to have been formulated as a general solution to the trickier types of spelling exceptions.)

A recoding procedure is properly part of the stem-

ming routine itself, although it introduces an element of iteration into it. Recoding occurs immediately following the removal of an ending and makes such changes at the end of the resultant stem as are necessary to allow the ultimate matching of varying stems. These changes may involve turning one stem into another (e.g., the rule *rpt* → *rb* changes *absorpt* to *absorb*), or changing *both* stems involved by either recoding their terminal consonants to some neutral element *(absorb* → *absorß, absorpt* → *absorß),* or removing some of these letters entirely, that is, changing them to nullity *(absorb* → *absor, absorpt* → *absor).*

In proposing a recoding procedure, one makes the assumption that most of the spelling changes that occur can be adequately covered by a small set of context-sensitive transformational rules—that the exceptions are predictable enough so that the number of "accidental" transformations is not sufficiently great to distort the whole stemming system. An example of such an accidental transformation is *send* → *sens,* generated by the rule *end* → *ens.* This rule was originally intended to take care of such pairs as *extend:extensive,* but instead it has made the stem *sens* ambiguous (it now stands for both *send* and *sense).* Fortunately the ambiguity can be resolved by changing the rule to "*end* → *ens* except following *s*"*;* but this type of solution may not be possible in all cases.

This assumption of a large amount of regularity in spelling changes appears to be a sound one. However, the exceptions are not *totally* predictable (i.e., not always dependent on immediate orthographic context); therefore a certain number of mistakes will result, which must be balanced against the favorable attributes of the method, like its speed.

It is important to note that the rules used in recoding should be not only context-sensitive but also *ordered.* Suppose we have the two rules:

1. Remove one of double *b, d, g, m, n, p, r, s, t.*
2. Turn terminal *d, r, t, z* into *s.*

The second rule is intended to take care of *collide: collision,* etc. Now suppose we have the words *admittance* and *admission.* The first is stemmed to *admitt,* the second to *admiss.* If the rules are applied in the order given, *admitt* → *admit* → *admis* and *admiss* → *admis;* if they were reordered, however, the result would be *admitt* → *admits, admiss* → *admis,* which is incorrect.

A more complete set of recoding rules of the type exemplified above is given in Appendix C. These rules are subject to revision, of course; it would also be desirable to contrast their results with those produced by neutralizing or nullifying transformations (see above).

The second kind of cure for spelling exceptions, partial matching, is methodologically quite different from recoding. Yet the basic assumptions, and the results, may be similar. The first assumption is that spelling changes in English are restricted to certain types which *may* occur, but do not *always* occur. The second assumption is that these changes involve no more than two letters at the end of a stem--this is merely an empirical result which has not yet been contradicted. It has also been observed that the sequences of letters that cause difficulty are often common to more than one class of exceptions. In recoding, this means that some rules can cover more than one type of exception, although it is not usually the case.

The crucial difference between recoding and partial matching is this: a recoding procedure is part of the stemming algorithm while a partial-matching procedure is not. Partial matching operates on the output from the stemming routine at the point where the stems derived from catalogue terms are being searched for matches to the user's stemmed query. All *partial* matches, within certain limits, are retrieved rather than just all *perfect* matches; discrepancies are resolved after retrieval, not in the previous stemming procedure. This has the advantages of reducing stemming to the one-step process of removing an ending and of eliminating the context specifications sometimes needed in recoding. The disadvantages, which are not so obvious, can be discussed only after a more complete description of a partial-matching procedure is given.

Such a procedure starts with an unmodified stem S1 —again, *absorpt* is a good example. The first step is to search the list of stemmed catalogue terms for all those which begin with S_1 minus its last two letters: in this case, all stems of any length beginning with *absor,* which we call S_2. Of course, special provisions will have to be made for cases in which S_1 is only two or three letters long. Among those stems returned will be *absorpt* and *absorb. Absorbefaci,* the stem of *absorbefacient,* may also be found. This last item will be eliminated, probably for the better, by the next step of the procedure, which discards all stems more than two characters longer than S_1 (here, more than nine letters long). We then have collected all stems which match *absorpt* within two letters in either direction. Given any one of these, S_j, a *final* match is allowed between S_j and S_1 if and only if either $S_j = S_1$ or the following conditions are satisfied:

1. The stems S_j and S_1 must match at least up to two letters before the end of the longer of them.
2. If S_j and S_1 are the same length and differ by one letter, this letter plus a blank must occur on a closed list (see Appendix D) for each stem.
3. If S_j and S_1 are the same length and differ by two letters, each sequence of two letters must occur on the list.
4. If S_j and S_1 differ in length by one, the last two letters of the longer, and the last of the shorter plus a blank, must occur on the list.
5. If S_j and S_1 differ in length by two, the last two letters of the longer must occur on the list.

The above rules amount essentially to examining the last two letters of stems that match up to that point; if the stems are different lengths, all "missing letters"

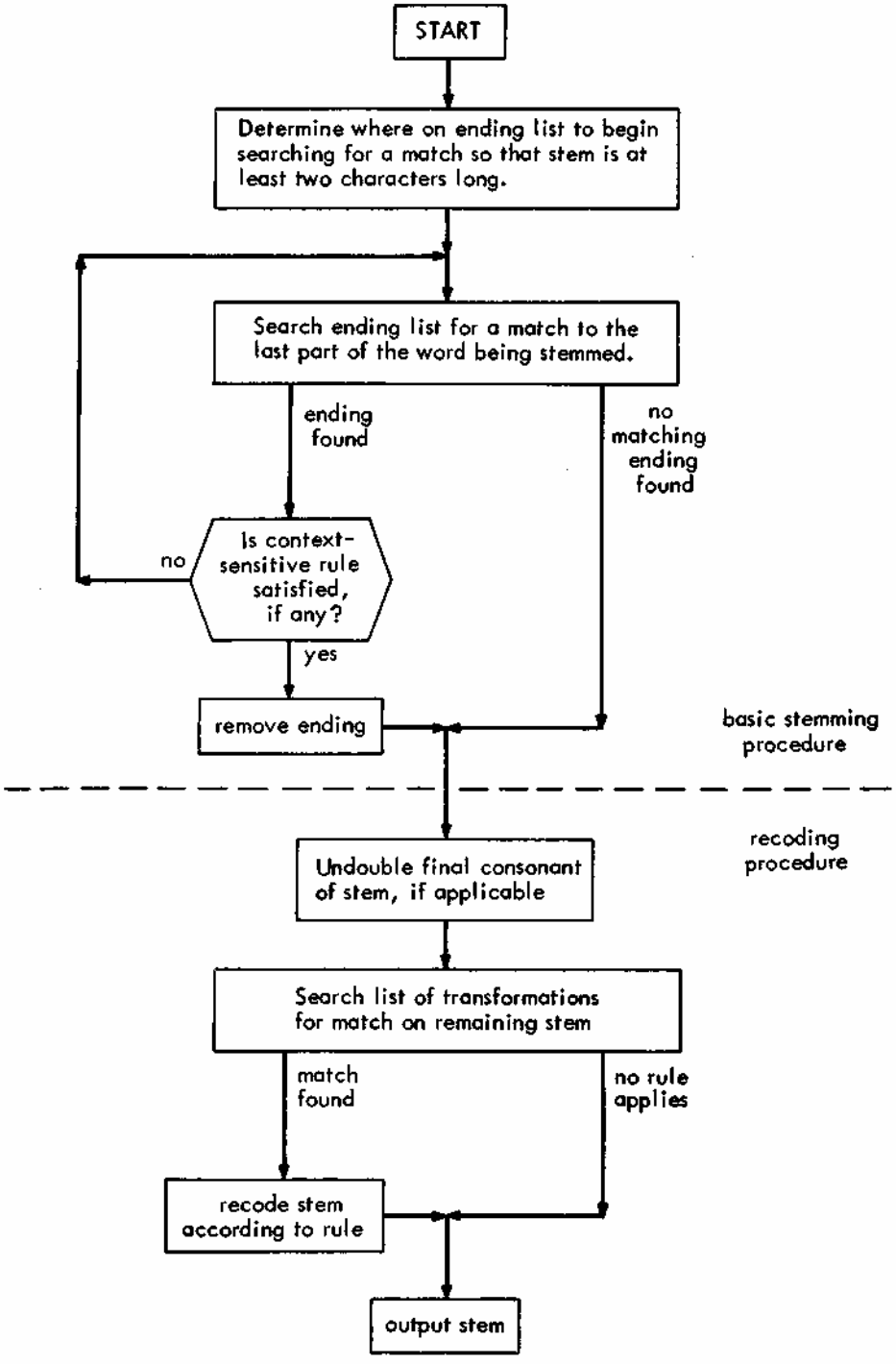

FIG. 1.—Stemming and recoding routines

Input	Initial Stem	Recoded Stem
magnesia	magnes	magnes
magnesite	magnesit	magnesit
magnesian	magnes	magnes
magnesium	magnesium	magnesium
magnet	magnet	magnes
magnetic	magnet	magnes
magneto	magnet	magnes
magnetically	magnet	magnes
magnetism	magnet	magnes
magnetite	magnetit	magnetit
magnetitic	magnetit	magnetit
magnetizable	magnet	magnes
magnetization	magnet	magnes
magnetize	magnet	magnes
magnetometer	magnetometer	magnetometer
magnetometric	magnetometr	magnetometr
magnetometry	magnetometr	magnetometr
magnetomotive	magnetomot	magnetomot
magneton	magneton	magneton
magnetostriction	magnetostrict	magnetostrict
magnetostrictive	magnetostrict	magnetostrict
magnetron	magnetron	magnetron
metal	met	met
metallic	met	met
metallically	metall	metall
metalliferous	metallifer	metallifer
metallize	metall	metall
metallurgical	metallurg	metallurg
metallurgy	metallurg	metallurg
induction	induct	induc
inductance	induct	induc
induced	induc	induc
angular	angul	angl
angle	angl	angl

FIG. 2.—Output from stemming routine

Input	Initial Stem	Recoded Stem
magnesia	magnes	magnes
magnesite	magnes	magnes
magnesian	magnes	magnes
magnesium	magnes	magnes
magnet	magnet	magnet
magnetic	magnet	magnet
magneto	magnet	magnet
magnetically	magnet	magnet
magnetism	magnet	magnet
magnetite	magnet	magnet
magnetitic	magnet	magnet
magnetizable	magnet	magnet
magnetization	magnet	magnet
magnetize	magnet	magnet
magnetometer	magnetometer	magnetometer
magnetometric	magnetometr	magnetometer
magnetometry	magnetometr	magnetometer
magnetomotive	magnetomot	magnetomot
magneton	magnet	magnet
magnetostriction	magnetostrict	magnetostrict
magnetostrictive	magnetostrict	magnetostrict
magnetron	magnetron	magnetron
metal	metal	metal
metallic	metall	metal
metallically	metall	metal
metalliferous	metallifer	metallifer
metallize	metall	metal
metallurgical	metallurg	metallurg
metallurgy	metallurg	metallurg
induction	induct	induc
inductance	induct	induc
induced	induc	induc
angular	angul	angl
angle	angl	angl

FIG. 3.—Output after revision of program

in the shorter are represented by blanks. The "closed list" needed for this routine is given in Appendix D.

It may appear that an unacceptable number of "wrong" matches would result from this procedure, since there are no restrictions on which pairs of items on the list may be used to produce a match. There are two defenses against this view:

First, such a closed list does exist. Many partial matches will *not* be allowed. Of those that are allowed erroneously, many would have been produced also by a recoding procedure, for much the same reasons.

Second, we can make a probabilistic argument. Most of the stems used will probably be fairly long—long enough so that there are unlikely to be many matches within two letters. Any S_j found by searching with S_2 stands a good chance of being related to S_2, and thus to S_1.

In short, while a partial-matching procedure may produce *no fewer wrong* matches than recoding, it will probably produce *more right* ones. It is inherently more flexible than recoding rules; all classes of exceptions do not have to be specified beforehand. Part of this flexibility results from allowing S_1 and S_j to differ in length by two letters in *either* direction. Yet this condition also provides a built-in barrier against certain types of wrong matches, as the following example illustrates:

Convex is recoded to convic by the rule *ex* → *ic; convict,* the stem of *conviction,* is recoded to *convic by* the rule *ct* → *c.* This erroneous match is *not* allowed in partial matching, since although condition (4) is satisfied, condition (1) is not.

Partial matching is a kind of controlled recoding; the recoding takes place *only* if a partial, but not complete, match is found. The original stem is still preserved, however, providing a constant check for violation of condition (1).

Using partial matching as a substitute for recoding does have one major disadvantage for a system using disk storage, as Intrex does, and it is a potentially serious one. In some cases, the time-consuming retrieval from the disk of a great number of partial matches, those beginning with S_2, will be necessary. These cases are most likely to occur with very short stems. The question is whether in such instances S_2 can be lengthened (made closer to S_1) enough to avoid this problem and still retrieve all acceptable matches. Empirical data are needed to answer this question, as well as to determine whether the number of short stems used is great enough to warrant concern. Any timing, programming, or other complications which partial matching introduces must be small enough to be balanced out by other advantages it may offer.

VI. The Two-Phase Stemming Routine and Its Results

Several progressively more advanced versions of the Intrex stemming routine have been coded in AED (a compiler language developed at the Electronic Systems Laboratory) [7, pp. *367-85]* and run on sample batches of words, using the MIT 7094 CTSS system. The flow chart in Figure 1 shows the most important features of the stemming and recoding parts of the program.

While a full evaluation of this stemming system within the Project Intrex environment will not be possible until the augmented catalogue data base is completed, output so far indicates that the procedures used are workable and will yield very good results with only minor changes. These changes involve the list of endings and occasionally the recoding rules; the types of operations performed remain the same.

To give some idea of the alterations that are needed to make the system highly effective, I shall discuss several of the changes that have been made in the program. Figure 2 shows the result of stemming several groups of related words. An obvious problem was that "magnet" and "magnesium" had the same recoded stem. This problem was easy to fix by changing recoding rule 32 from *et* → *es* to *et* → *es* except following *n.*

An additional recoding rule took care of the discrepancy between *meter*→ *meter* and *metric* → *metr:metr* → *meter.* All other changes involved the stemming procedure: *-ium, -ite,* and *-itic* were added to the list of endings, with the stipulation that *-ite* be removed only in certain rather limited cases and *-itic* only after *t* or *ll;* the rule governing *-al-* endings was changed so that they are not removed after *met-; l* was added to the list of stem-final consonants to be undoubled; and the context in which the removal of *-on* is allowable was broadened to include single *t.* The results after these changes are shown in Figure 3. It is expected that several more such evaluations of a random group-sample will catch most similar difficulties still left in the program, although it is likely that minor revisions will be required as long as the vocabulary of the data base continues to increase.

Appendix A

LIST OF ENDINGS*

.11.	.09.	.09.—*Cont.*	.08.
alistically B	allically C	entiality A	ableness A
arizability A	antaneous A	entialize A	arizable A
izationally B	antiality A	entiation A	entation A
	arisation A	ionalness A	entially A
.10.	arization A	istically A	eousness A
antialness A	ationally B	itousness A	ibleness A
arisations A	ativeness A	izability A	icalness A
arizations A	eableness E	izational A	ionalism A
entialness A	entations A		ionality A

.08.—*Cont.*	.06.—*Cont.*	.05.—*Cont.*	.03.
ionalize A	ialist A	inate A	acy A
iousness A	iality A	iness A	age B
izations A	ialize A	ingly B	aic A
lessness A	ically A	inism J	als BB
	icance A	inity CC	ant B
.07.	icians A	ional A	ars O
ability A	icists A	ioned A	ary F
aically A	ifully A	ished A	ata A
alistic B	ionals A	istic A	ate A
alities A	ionate D	ities A	eal Y
ariness E	ioning A	itous A	ear Y
aristic A	ionist A	ively A	ely E
arizing A	iously A	ivity A	ene E
ateness A	istics A	izers F	ent C
atingly A	izable E	izing F	ery E
ational B	lessly A	oidal A	ese A
atively A	nesses A	oides A	ful A
ativism A	oidism A	otide A	ial A
elihood E		ously A	ian A
encible A	.05.		ics A
entally A	acies A	.04.	ide L
entials A	acity A	able A	ied A
entiate A	aging B	ably A	ier A
entness A	aical A	ages B	ies P
fulness A	alist A	ally B	ily A
ibility A	alism B	ance B	ine M
icalism A	ality A	ancy B	ing N
icalist A	alize A	ants B	ion Q
icality A	allic BB	aric A	ish C
icalize A	anced B	arly K	ism B
ication G	ances B	ated I	ist A
icianry A	antic C	ates A	ite AA
ination A	arial A	atic B	ity A
ingness A	aries A	ator A	ium A
ionally A	arily A	ealy Y	ive A
isation A	arity B	edly E	ize F
ishness A	arize A	eful A	oid A
istical A	aroid A	eity A	one R
iteness A	ately A	ence A	ous A
iveness A	ating I	ency A	
ivistic A	ation B	ened E	.02.
ivities A	ative A	enly E	ae A
ization F	ators A	eous A	al BB
izement A	atory A	hood A	ar X
oidally A	ature E	ials A	as B
ousness A	early Y	ians A	ed E
	ehood A	ible A	en F
.06.	eless A	ibly A	es E
aceous A	elily A	ical A	ia A
acious B	ement A	ides L	ic A
action G	enced A	iers A	is A
alness A	ences A	iful A	ly B
ancial A	eness E	ines M	on S
ancies A	ening E	ings N	or T
ancing B	ental A	ions B	um U
ariser A	ented C	ious A	us V
arized A	ently A	isms B	yl R
arizer A	fully A	ists A	s' A
atable A	ially A	itic H	's A
ations B	icant A	ized F	
atives A	ician A	izer F	.01.
eature Z	icide A	less A	a A
efully A	icism A	lily A	e A
encies A	icist A	ness A	i A
encing A	icity A	ogen A	o A
ential A	idine I	ward A	s W
enting C	iedly A	wise A	y B
entist A	ihood A	*ying* B	
eously A		yish A	

* The capital letters after each letter-group are a condition code, not part of the ending itself. For key, see Appendix B.

Appendix B

CONDITION CODES FOR CONTEXT-SENSITIVE RULES ASSOCIATED WITH CERTAIN ENDINGS

A . . . No restrictions on stem
B . . . Minimum stem length = 3
C . . . Minimum stem length = 4
D . . . Minimum stem length = 5
E . . . Do not remove ending after *e*
F . . . Minimum stem length = 3 and do not remove ending after *e*
G . . . Minimum stem length = 3 and remove ending only after *f*
H . . . Remove stem ending only after *t* or *ll*
I . . . Do not remove ending after *o* or *e*
J . . . Do not remove ending after *a* or *e*
K . . . Minimum stem length = 3 and remove ending only after *l*, *i*, or *uae* (where *a* stands for any letter)
L . . . Do not remove ending after *u*, *x*, or *s*, unless *s* follows *o*
M . . . Do not remove ending after *a*, *c*, *e*, or *m*
N . . . Minimum stem length = 4 after *sa*, elsewhere = 3
O . . . Remove ending only after *l* or *i*
P . . . Do not remove ending after *c*
Q . . . Minimum stem length = 3 and do not remove ending after *l* or *n*
R . . . Remove ending only after *n* or *r*
S . . . Remove ending only after *dr* or *t*, unless *t* follows *t*
T . . . Remove ending only after *s* or *t*, unless *t* follows *o*
U . . . Remove ending only after *l*, *m*, *n*, or *r*
V . . . Remove ending only after *c*
W . . . Do not remove ending after *s* or *u*
X . . . Remove ending only after *l*, *i*, or *uae*
Y . . . Remove ending only after *in*
Z . . . Do not remove ending after *f*
AA . . . Remove ending only after *d*, *f*, *ph*, *th*, *l*, *er*, *or*, *es*, or *t*
BB . . . Minimum stem length = 3 and do not remove ending after *met* or *ryst*
CC . . . Remove ending only after *l*

Appendix C

TRANSFORMATIONAL RULES USED IN RECODING STEM TERMINATIONS

1 . . . Remove one of double *b*, *d*, *g*, *l*, *m*, *n*, *p*, *r*, *s*, *t*
2 . . . iev → ief
3 . . . uct → uc
4 . . . umpt → um
5 . . . rpt → rb
6 . . . urs → ur
7 . . . istr → ister
7*a* . . metr → meter
8 . . . olv → olut
9 . . . ul → l except following *a*, *i*, *o*
10 . . . bex → bic
11 . . . dex → dic
12 . . . pex → pic
13 . . . tex → tic
14 . . . ax → ac
15 . . . ex → ec
16 . . . ix → ic
17 . . . lux → luc
18 . . . uad → uas
19 . . . vad → vas
20 . . . cid → cis
21 . . . lid → lis
22 . . . erid → eris
23 . . . pand → pans
24 . . . end → ens except following *s*
25 . . . ond → ons
26 . . . lud → lus
27 . . . rud → rus
28 . . . her → hes except following *p*, *t*
29 . . . mit → mis
30 . . . end → ens except following *m*
31 . . . ert → ers
32 . . . et → es except following *n*
33 . . . yt → ys
34 . . . yz → ys

Appendix D

STEM TERMINATIONS USED IN A PARTIAL-MATCHING PROCEDURE

ββ	er
bβ	es
cβ	ex
dβ	gg
gβ	ic
lβ	il
nβ	is
pβ	nn
rβ	or
sβ	os
tβ	ot
vβ	pp
xβ	pt
zβ	ss
bb	tt
ct	ul
dd	ut

NOTE.—ß stands for a blank. Stems are assumed to occur in a field of blanks.

Received October, 1967
Revised November, 1968

References

1. Dyson, G. M. "Computer Input and the Semantic Organization of Scientific Terms." *Information Storage and Retrieval* (April 1967), pp. 35-115.
2. Earl, Lois L. "Part-of-Speech Implications of Affixes." *Mechanical Translation and Computational Linguistics,* vol. 9, no. 2 (June 1966).
3. Earl, Lois L. "Structural Definition of Affixes from Multisyllable Words." *Mechanical Translation and Computational Linguistics,* vol. 9, no. 2 (June 1966).
4. Lejnieks, Valdis. "The System of English Suffixes." *Linguistics* 29 (February 1967):80-104.
5. Overhage, Carl F. J. "Plans for Project Intrex." *Science* 152 (May 20, 1966): 1032-37.

6. Resnikoff, H. L., and Dolby, J. L. "The Nature of Affixing in Written English." Part I: *Mechanical Translation and Computational Linguistics,* vol. 8, no. 3 (June 1965); Part II: *Mechanical Translation and Computational Linguistics,* vol. 9, no. 2 (June 1966).
7. Ross, Douglas T. "The Automated Engineering Design (AED) Approach to Generalized Computer-aided Design." *Proceedings of the 22d National Conference, Association for Computing Machinery.* Washington, D.C.: Thompson Book Co., 1967.
8. *Normal and Reverse Word List.* Compiled under the direction of A. F. Brown at the University of Pennsylvania, under a contract with the Air Force Office of Scientific Research (AF 49 [638]-1042), Department of Linguistics, Philadelphia, 1963.
9. Salton, Gerard. *Automatic Information Organization and Retrieval.* New York: McGraw-Hill, 1968.
10. Salton, Gerard, and Lesk, M. E. "The SMART Automatic Document Retrieval System." *Communications of the ACM,* vol. 8, no. 6 (June 1965).