to be printed. These vectors are needed, because, for some search-and-print modes, the decision to print cannot be made when a node is found. When a decision to print is finally made, these vectors provide the necessary print information, and rereadings of the VOCAB and DRCTRY are eliminated. These vectors are organized so as to make their entries correspond with the entries in PATH. Thus, WORDSP(I) = VWORD(PATH(I)) and CATSP(I) = DCAT(PATH(I)). As a consequence, half of each -SP vector is unused, a small expense of memory to save the time that would otherwise be needed to compute their indices: only one index serves for all three vectors.

Other data items of interest are: CURCAT, the index in DRCTRY of the first category. This location marks the starting point of a word ring search; PRINTNDX, which indicates the last position in PATH (and hence in WORDSP and CATSP) that has been printed.

B. PREFIX

by
John B. Smith

BACKGROUND

Project VIA's need of a computational procedure for determining the presence of an English prefix on a word has both immediate and far-reaching implications.

In order to determine patterns of inter-relations among content carrying words, it early became apparent that procedures would have to be developed that could "recognize" or group together words with the same root or stem. Part of this task was accomplished by SUFFIX, which groups together words of the same root form but with different suffixes. PREFIX accomplishes the other half of the task. It allows us to note the presence of a concept or idea carried in the root of the word but modified and masked by the prefix. Thus it has immediate use in the VIA package.

Although syntactic analysis is of no immediate concern for VIA, recent computational studies have indicated the importance of affixes as indicators of part-of-speech. This consideration led to Resnikoff's and Dolby's work on operational definitions of affixes and an algorithmic approach to determining affixes.* Their work has been followed up by Lois Earl in her attempts to assign part-of-speech categories by rules based primarily on affixes and internal vowel clusters.** Unfortunately, her goal of 95% accuracy has been attained only hypothetically because of errors in her dictionary, and her work is restricted to a corpus of only some 20,000 words. PREFIX, on the other hand, is defined over a considerably larger corpus, the unabridged *Random House Dictionary*.

*H.L. Resnikoff and J.L. Dolby, "The Nature of Affixing in Written English," *Mechanical Translation*, VIII (1965), 84-89. Also "The Nature of Affixing in Written English Part II," *Mechanical Translation*, IX (1966), 23-33.

*Lois L. Earl, "Automatic Determination of Parts of Speech of English Words," *Mechanical Translation*, X (1967), 53-67.

Consequently, PREFIX may have important implications in syntactic studies that lie outside the immediate concerns of Project VIA.

GENERAL APPROACH: Essentially, PREFIX's approach is a table look-up procedure, but without the disadvantage of costly time consumption of multiple searches through the entire table. An extensive list of admissible English prefixes was complied by consulting available lists of affixes and by consulting our working dictionary. We placed two linguistic restrictions on prefixes:

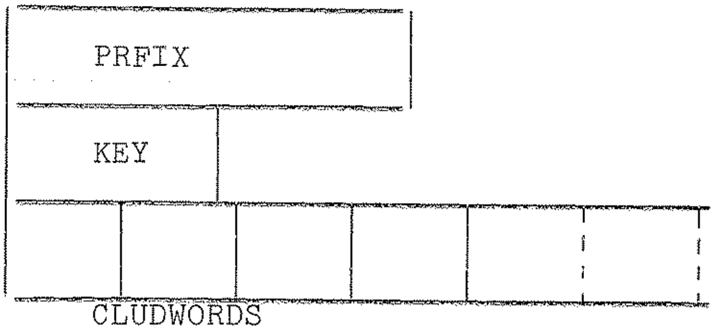
1. The prefix must be a bound morpheme.
2. A word is considered to have a prefix only if the remainder of the word, without the prefix, is independent, i.e., not a bound morpheme.

After preparing our list of prefixes, we next had to account for words whose initial letters are identical with given prefixes but which are not prefix-carrying words. For example, *at*, although beginning with *a*, is not a prefix-carrying word; *atypical* would be. SUFFIX functions by having lists of exceptions. However, we found such an approach impractical for many prefixes. The *a* prefix is an example of this problem: an exception list would involve most of the words beginning with *a* listed in our dictionary. One solution to the problem is to use an inclusion list and consider only those words on the list as having legitimate prefixes. Such an approach would work well for the prefix *a*, but not for *in*. Our ultimate solution was to compile either an exception list or an inclusion list for each prefix, dependent upon which list would have fewer members. A note on problems of specific work selection will be given later in this paper.

PREFIX: As pointed out above, PREFIX is a table look up procedure; however, since text input is assumed to be in logical records, one word per record, and the records to be in alphabetical order, the look up time can be reduced to a minimum. In fact, the task can be accomplished with just one complete pass through the prefix lists. Each prefix is loaded into a PL/I structure along with its accompanying list of words and the key that specifies whether the list is an inclusion list or an exclusion list. This structure has the following format:

```
91  PTABLE (35),
    02  PRFIX CHARACTER (8),
    02  KEY FIXED DECIMAL (1),
    02  CLUDWD (300) CHARACTER (18);
```

or



for one prefix with accompanying CLUD list. The structure, PTABLE, will accommodate 35 prefixes, each with as many as 300 accompanying words.

Since there are obviously more than 35 prefixes in the English language, we had to resort to an overlay approach to "roll in" and "roll out" the appropriate prefix lists. This task is performed by a call to a subroutine called PFETCH.

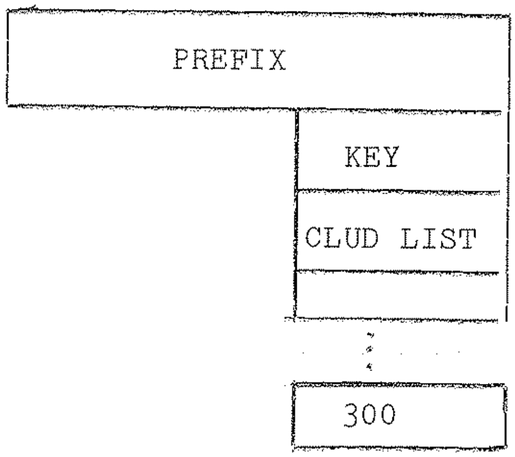
PFETCH: This subroutine reads a sequential data set of prefixes with accompanying lists--hence referred to as CLUD lists or CLUD words--and loads them into the structure PTABLE. This is done for all prefixes beginning with the same letter of the alphabet. When a prefix is read in that begins with a different letter, it is stored temporarily, and execution falls into some "housekeeping" tasks which will be explained later. Control then passes back to the main procedure. For example, the first call to PFETCH will load in all _a_ prefixes, with CLUD lists, until the first prefix beginning with a _b_ is read. Prefixes, like text-word records, are in alphabetical order as are their CLUD lists.

MAIN PROCEDURE: The main procedure is controlled by a large DO-loop for which each value of the indexing variable represents a letter of the alphabet. Incoming text records are first tested against the control letter of the alphabet. Processing continues so long as the first letter of a text word matches the control letter; if not, PFETCH is called to load in the next group of prefixes. [The text word is next checked to see if it is identical with the preceding word that was just processed. If so, it is either processed or rejected as was the preceding word. If the word is different then it falls into a series of tests.]

First the word is tested to determine its length. If the word has fewer than four characters, it is rejected (REJECT is set equal to the word so that the next word read in can be tested against it). This is done on the assumption that words with three and fewer characters do not contain admissible prefixes. We have not found exceptions to this rule in any tests yet processed.

If the word is longer than three characters it is tested against the list of prefixes. If the prefix is of length N, the first N letters of the word are checked for a match. If these conform, then a check of the accompanying CLUD list is performed. The word is checked against the words of the CLUD list until a match is found or the word is no longer further along in alphabetical sequence than the remaining words in the CLUD list.

PTABLE: for each prefix.



If the word is found to match one of the words in the CLUD list, then the prefix key is consulted. If the key is 0--indicating an exclusion list--the word is rejected, REJECT is set equal to the word, and a new word is read in for testing. If the key is 1--indicating an inclusion list--a duplicate record, except for the omission of the prefix, is created. LSTWORD is set equal to the word indicating that a valid prefix was found for subsequent testing, and a new text word is read in. When a prefix match is found, the location of the prefix within the PTABLE structure is noted, and similarly for a match within the CLUD list. Since the text words, prefixes, and CLUD lists are all in alphabetical order, subsequent tests for text words can begin with the prefix and CLUD word last found to match a text word. The prefixes and CLUD lists are processed in their entirety only once, thus greatly reducing look up time. The time gained, however, by passing through the list of prefixes only once is not without some qualifications.

This last point can best be developed by an illustration. The word atypical contains a legitimate a prefix, but in alphabetical sequence it would come after words with ab prefixes, ad prefixes, etc. If we wish not to keep searching the prefix lists, prefixes that admit such words must be flagged. It turns out that each such prefix is "contained in" the prefix immediately following it. That is, the "troublesome" prefix will be shorter in length than the succeeding prefix and will match it letter-for-letter for its length. Some such prefixes are *a* (contained in *ab*), *arch* (contained in *arche*), etc. The task of flagging each such prefix is performed in PFETCH. The locations in PTABLE of all prefixes of this kind are loaded into an array called PERMFIX, with space for ten prefixes (actually what is stored is a number pointing to the location of the prefix in PTABLE--thus for *a* the pointer would be '1').

It will be recalled that we tested each text word for a match with a prefix of N letters. When a match was found, the prefix was marked and subsequent testing began there. If the prefix does not match the first N letters of a word, a test is made to see if the word follows the prefix in alphabetical sequence. For example, if testing for the word *aftermath* begins with the prefix *ad*, a mismatch of the first two letters with the prefix will occur. *Aftermath* will then be seen to come after *ad* in alphabetical sequence; consequently control will shift to the next prefix, and so on until a match is found or the word precedes the prefix in sequence. At this point testing will shift to the group of prefixes that admit words in later sequence than words with the next lower prefix--as was the case with *atypical*. The word is tested against all such prefixes--referenced through the pointers in PERMFIX-- and their associated CLUD lists. If a match of both prefix and CLUD word is found, a duplicate record is formed or not depending upon the key. If a match of prefix but not CLUD word is found, a duplicate record is formed if the key is 0 (indicating that the list is an exclusion list). Either

REJECT or LSTWORD is set equal to the word accordingly.

PRINT: PRINT is a subprocedure that does the actual processing of the prefix. In the present experimental version of PREFIX the prefix is lost; however, in the functioning version it will remain as a separate entry within the logical record for each text word. PRINT is called whenever an additional record is to be created. Into the sequential data set is introduced a duplicate record but with the word stripped of prefix. A listing on the printer is also made for manual reference. Format is identical to input format and is as follows:

1	3	9	12	19
2	6	3	7	18
↑ LENGTH OF RECORD	LIN.#	PG.	BLANK	WORD

The actual removal of only the prefix is accomplished by using the VARYING character attribute of PL/I. By storing the prefix in a location with this attribute, the computer records the actual length of the record contained (in this case the prefix). Consequently, the portion of the text word without the prefix can be picked off by using the SUBSTRING operator. The second operand, the position of the variable (in this case the text word) at which the substring is to begin, is set equal to the length of the particular prefix plus 1. After each call to PRINT, processing continues as before.

TABLE PREPARATION: Scientific and obsolete words, proper nouns, and multiple word idioms are not included in the CLUD lists; however, words marked "archaic" that might appear in literary texts are included. The problem of accounting for forms of words to be included but with variant suffix forms was solved in the following way. Once we determined that a root form was to be included in the list, we made the entry conform to only those letters that the variant forms share in common. Thus the CLUD list entry for *complete*, *completely*, *completing*, etc. would consist of the letters *complet*. This approach is applicable only when the entry form excludes all words not of the same root and which are not to be included in the CLUD list. This constraint necessitated our marking certain short words as complete in themselves. For example, *add* is included in the form 'ADD '; otherwise, the program would assume that the *add* entry would include all words with these first three letters.

At present, the program is operational, but we are in the process of making corrections and additions to our CLUD lists to account for unforeseen omissions and inclusions. One of our working hypotheses is that the prefix is much more fundamentally involved with the semantic content of a word than the suffix; but it also appears much less frequently than the suffix within English texts. However, our experience with PREFIX is limited and initial assumptions may well be modified later.