The word has now had all its initial prefixes removed. All of its suffixes have also been removed, and none of these resulted in a break to the left of the optimum point. The procedure \texttt{vcscan} searches for one acceptable break in the whole of the remaining word.

Starting at the optimum point, the procedure locates the V/C pattern encompassing it, and resets \texttt{back}. It uses \texttt{nextpattern} to locate the next pattern to the left of \texttt{back} of the form VCV, VCCV, VCCCCV, CVC, CVVC, CVVVVC. If it finds none, or the front of the current sub-word is reached, then it tells \texttt{vcscan} to exit. If one of the above patterns was found, it accesses the corresponding table and this is searched to find a match with the actual letters.

If no match is found by \texttt{search}, then \texttt{vcscan} checks for acceptable suffixes. If none is found, \texttt{vcscan} repeats, moving leftwards.

When a match is found, the break point indicated is tested against the optimum point. If the break is to the right of the optimum point, \texttt{vcscan} continues its searching leftwards. If the break is to the left, the \texttt{back} of the current sub-word is recalculated. \texttt{vcscan} then returns a positive value.

\begin{verbatim}
procedure vcscan (var result: Integer);
{Compiled version occupies 2160 bytes of memory}
{Used by: afrhyp, wi with parameter as defined above}
{Uses procedures: search, suffixscan}
{Globals used: table, front, back, hword, hopt, hcv, vctable.}
{Returns: 0 if no break found to left of optimum point, or word short
+1 if acceptable break found left of optimum point}
var i : integer; {used to return result of nextpattern}
procedure reset_back;
{starts at optimum point and searches forward for a letter with opposite c/v value.}
{Stops if back reached. If back not reached, back is reset to this letter.}
var finish : boolean;
 i : integer;
begin
 finish := false;
i := hopt;
 repeat
 if hcv(hopt) = hcv(i)
 then begin i := i + 1; if i > back then finish := true end
 else begin back := i; finish := true end
 until finish;
 end; (reset_back)

procedure nextpattern (var j : integer);
{starts at back and searches for next cv pattern}
{Returns 0 if none found before front; -1 if " found; 1 if VCV or CVC found, 2 if VCCV or CVVC, n for the number of characters in the enclosed string.}
end; (vcscan)
\end{verbatim}
{back is set to the last character of the string.}

var finish : boolean;
i : integer;

begin
finish := false;
i := back - 1;
repeat
if (hcv[i] = -1) and (hword[i] = "") then begin back := i - 1; j := -1; finish := true end
else if hcv[i] = hcv[back] then begin back := i + 1; if back - front < 2 then j := 0 else j := 1; finish := true end
else begin i := i - 1; if i < front then begin j := 0; finish := true end else j := 1 end
until finish;
finish := false;
if j = 1 then repeat
if (hcv[i] = -1) and (hword[i] = "") then begin back := i - 1; j := -1; finish := true end
else if hcv[i] = hcv[back] then begin j := back - i - 1; finish := true end
else begin i := i - 1; if i < front then begin j := 0; finish := true end end
until finish;
end; (nextpattern)

procedure testsuffixes;
var found : integer;
begin
repeat suffixscan(found) until found >= 0;
if found = 0 then result := -1 else result := +1
end; (testsuffixes)

procedure paircheck;
var found : integer,
i : integer,
datum : integer;
begin
if hcv[back] = 0 (VCCV) then i := table.vetable[1] else i := table.vetable[3]; [CVVC]
datum := back - 2;
search(i,datum,found);
back := back - found;
if back <= hopt then result := +1 else testsuffixes
end; (paircheck)

procedure triplecheck;
var f: char : integer,
i : integer,
datum : integer;
begin
if hcv[back] = 0 (VCCCV) then i := table.vetable[7] else i := table.vetable[4]; [CVVVC]
datum := back - 3;
search(i,datum,found);
if found = 0 then paircheck
else begin back := back - found; if back <= hopt then result := +1 else testsuffixes end;
end; (triplecheck)

begin
reset_back;
end;
8.5 Search Procedure

This is a procedure used by `prefixscan`, `suffixscan`, and `vcescan`, and is a generalised table search. The tables must be set up in the format described below, and the procedure is called with two parameters, the address of the table and the search argument i.e. the word portion whose match is sought. It returns the answer 'not found', or a suitable code which must then be interpreted by the calling procedure, but typically will be the number of letters which were matched.

The procedure permits both positive and negative comparison, together with logical AND and OR. The meanings of characters in the tables are given below.

Valid in both positive and negative strings

Letters A to Z

Accents " and @

/ represents OR
0 represents any letter
9 represents END OF RECORD
* represents END OF TABLE
Valid in positive strings only
- represents BEGIN NEGATIVE COMPARISON
  1 to 8 represents SHIFT DATUM LEFT
< represents CONSONANT
= represents VOWEL
: represents DOUBLE LETTER

Valid in negative strings only
+ represents BEGIN POSITIVE COMPARISON

At the start of each new record, comparison is assumed to be positive.

Examples of this coding are given below with their interpretation.

P-Q9 P not followed by Q
P/Q9 r or Q
P/Q-R/ST9 P or Q not followed by RT or by ST
<1-P9 any consonant, but not P
PQ0-R+S9 PQ followed by any letter, followed by anything but R, then followed by S

The search procedure requires as parameters a table to be searched, and a position in word where string matching is to begin. It returns the result 0 if no match is found in the table, but non-zero if it detects a match. In this case the value returned will be the character following the end of record which contained the match in the table.

The procedure operates as follows. Each character of the table is tested against the target string until either a match is found, or the table is exhausted.

At the end of each record, two events can take place: the string is matched or the string is unmatched. Whether these provide valid breaks depends on whether positive or negative searching has taken place. There are two procedures to check this, match and mismatch. As soon as a match is found for one character while searching in positive mode, the flag pmatch is set. Likewise for mismatch in negative mode. If, at the end of a record, the positive flag has been set and the negative one remains unset, then the record matches the string and the search can return a found result, i.e. the character following the end-of-record code.
If, at the end of a record, any other combination of flags occurs, then the record does not match the string and the search continues to the next record. All the parameters must be reset, in particular the pointer to the start of the target string.

```pascal
procedure search (a: index; c: integer; var result: integer);

(Compiled version occupies 1582 bytes of memory)
(Uses procedures: none)
(Uses globals: hword, hend, hcv, table.
[Assumes the use of ASCII code]

var wordpt : integer;
tablept : index;
endsearch : boolean;
pmatch : integer;
mmatch : integer;
mode : char;

procedure match;
begin
if mode = '+' then pmatch := 1 else mmatch := 1;
wordpt := wordpt + 1;
if wordpt > hend + 1 then endsearch := true;
end;

procedure mismatch;
begin
with table do begin
if data[tablept] <> '9' then tablept := tablept + 1;
end;
end;

begin
with table do begin
result := 0;
tablept := a;
endsearch := false;
pmatch := 0;
mmatch := 0;
mode := '+';
wordpt := c;
end;
```
8.6 Exception Dictionary

As we have said earlier, we anticipate that the rules discussed here will still not be adequate in some cases. The methodology we have chosen allows unlimited coding of 'non-conforming' words in the appropriate tables. However, it may not be desirable to encode all exceptions like this since there may be local variances, commonly called House Rules. Thus the proper name Botha may be placed in an exception dictionary as BOTHA or BO-THA depending on the local requirements and standards. Imported words such as CAMOUFLAGE and MANOEUVRE can be placed in the Afrikaans dictionary as CA-MOU-FLAGE and MA-NOEU-VRE.

There is no way homonyms can be correctly identified using only orthographic rules, and perhaps the best way therefore of handling the few that exist is to place them in the dictionary without a break so that, if necessary, a manual decision still remains available.

The most frequent words in the dictionary are likely to be those which contain natural hyphens. Thus we could store OP-MEKAAR-VOLGENDE as OP-ME-KAAR-VOL-GENDE.
The exception dictionary will best be used at the stage between the normalisation of the original word and the entry to the prefix searching. It is suggested that a user-defined switch be made available in order to turn the dictionary searching on and off.
Chapter 9 SUPPORT ENVIRONMENT

9.1 Driver Program

This program is designed to provide a normal working environment for the \texttt{afrhyp} procedure to be tested.

The program first initializes the \texttt{afrhyp} procedure allowing it to set up its internal tables. The file of input words is then opened, and a suitable empty output file created. Into this will be written the hyphenated words. Each word is read and processed until the end of the file is reached, when both files are closed. The output file may then be printed in a separate operation.

The processing takes the form of locating all the valid hyphenation points in the word by moving the Optimal Point from the beginning to the end of the word. After each call to the \texttt{afrhyp} procedure, if a hyphenation point was found it is tested against the previous one to see if a new point has been located. If so, it too is stored in an array set up for this purpose.

When the whole word has been attempted in this manner, the hyphens are inserted in the word itself, starting from the end and working backwards. This is to avoid recalculating the pointers to the earlier hyphenation points.

Input Data

The file is called \texttt{AFRW ORDS} and will consist of Afrikaans words arranged one to a line, each ending with a carriage return code. The maximum length of a word is 64 characters; any letters in addition are ignored. Letters may be in upper or lower case. Diacritics in the input words are represented as a separate character before the character which they accent. Our representation for diaeresis is " (ASCII 172), circumflex is " (ASCII 94), grave is " (ASCII 96), and acute " (ASCII 171).
Example of file:
afrikaans
tekening
Woordlyste
grens
beroep
lidmaatskap
sien
láat
nimmer
enige
aarde
ontvangst
gesterf
opstanding
gemeenskap
geneeskunde
sielkunde
afgevaardigde
wêreld
reên

Output Data

The file is called HYPWORDS and will consist of Afrikaans words arranged one to a line, each ending with a carriage return code (ASCII 13). Letters may be in upper or lower case. Diacritics are represented as before. The inserted hyphen is represented by - (ASCII 45).

Example of file:
af-ri-kaans
tek-nings
Woord-lyst-te
gren-se
beroep
lid-maat-skap
sien
láat
nimmer
enige
aarde

60
ont-vang
ge-sterf
op-stan-ding
ge-meer-skap
genees-kun-de
siel-kun-de
af-ge-vaar-dig-de
wë-reid
roën
ont-vang
ge-sterf
op-stan-ding
ge-meen-skap
gé-nees-kun-de
siel-kun-de
af-ge-vaar-dig-de
wé-reld
reēn
Initialise AFRHYP
Open Input and Output Files
Read Word
Set up parameters

AFRHYP

Hyphen found
Yes
Store hyphen in list

End of word
Yes
Insert all hyphens in word
File Result

More Words
Yes

Next character position

this an account

No
Yes

No

End

Figure 9.1 Flowchart of the driver program