CMU-CS-78-134

A RECURSIVE SEGMENTATION PROCEDURE FOR CONTINUOUS SPEECH

G. Gill, H. Goldberg, R. Reddy, & B. Yegnanarayana

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

May 1978

ABSTRACT

The first step in the recognition of continuous speech by machine is segmentation of the utterance. The Harpy continuous speech recognition system, developed at Carnegie-Mellon University, uses a segmentation procedure based on simple time domain parameters called ZAPDASH. In this paper the parameters and the decision rules used in the segmentation are described. Considerations in the choice of parameters are discussed briefly. The heuristics used in arriving at some of the decision rules are also discussed. The performance of the segmentation scheme is evaluated by comparing the results with the results of hand segmentation of the waveform of the utterance. The results show an overall error rate of 4% in 34 utterances. However, even the error rate does not affect the recognition accuracy of the Harpy system significantly because the scheme is designed to provide several extra segments at the cost of speed of operation. The average duration of the segments obtained by this technique was found to be 4.7 centiseconds. The robustness of the segmentation scheme for noise and distortion in input speech is currently being investigated.

1. INTRODUCTION TO THE SEGMENTATION PROBLEM

Recognition of continuous speech [1] involves identification of the given utterance as one of the several possible (usually finite) classes by pattern matching. This requires determining the patterns for different classes and then comparing the text pattern with each one of them.

The first step in the recognition process is to break the given utterance into different sound classes. Unlike isolated word utterances, it is difficult to determine in continuous speech where one word ends and another begins. Even if the classes chosen are the linguistic units of speech called phonemes (instead of words), it is a nontrivial problem to identify the phoneme boundaries from continuous speech. This is because (1) phonemes are not uniquely defined for a given language [2], (2) there is no one-to-one relationship between the acoustic waveform and phoneme [3], (3) there are significant variations in different repetitions of the same phoneme by a single speaker and from speaker-to-speaker, (4) the characteristics of the acoustic patterns of phonemes exhibit much greater variability, depending on the context, and (5) the problem is complicated by noise and distortion in the input speech signal [4].

One way to overcome the difficulties in segmentation is to settle for a less stringent requirement. This may involve determination of boundaries associated with significant changes in the acoustic characteristics of speech. The hope is that, once such acoustic boundaries are determined, the segments between the boundaries can be assumed to have stationary properties. The assumption may not always be valid since the parameters of speech vary continually and rather abruptly sometimes. The parameters are usually estimated by performing frame by frame analysis of speech. Such an analysis makes it difficult sometimes to determine important acoustic boundaries occuring at abrupt changes in the parameters. Moreover, the distortions and variations in speech sounds further complicate the identification of acoustic boundaries through parametric extraction.

From the above discussion it is clear that unambiguous segmentation, in an absolute sense, either into phonemes or uniform acoustic segments or any other classes is not possible. For a given speech processing system the purpose and specific requirements of the segmentation process need to be defined in order to devise a suitable scheme. The objective of this paper is to describe the segmentation procedure adopted in the Harpy continuous speech recognition system [5]. The Harpy system operates on the basis that the continuous speech is broken into acoustically uniform segments. Reference templates are created from the segments of a training data using the spectral properties of the segments. Each of the segments of the test utterance are compared with the reference templates by spectral matching. The larger the number of segments the better it is for the accuracy of the system.

It is desirable to have a segmentation procedure that generates the smallest number of segments for a given utterance. This will help in speeding up the recognition process. The main purpose of segmentation, however, is to avoid labeling each analysis frame. Segmentation will enable labeling generally the uniform regions which represent the stationary spectral characteristics. Thus inaccuracies in the labeling at the boundaries, where the spectral features are likely to change rapidly, are minimized.

2. CHOICE OF PARAMETERS

2.1. Basis for the choice

There were several attempts [6]-[14] to segment continuous speech into phonemic units with the understanding that some phonemic boundaries may be missed and some acoustic boundaries may be inserted where there are no phonemic boundaries. As discussed earlier, such a result is to be expected due to contextual or coarticulation effects as well as variations and distortions in speech signals. Another problem with segmentation is devising an appropriate acoustic similarity measure which indicates a boundary if and only if there is a significant change in acoustic characteristics. Several segmentation techniques were proposed [6]-[14] using amplitude (or energy), zero-crossing and parameters representing spectral features.

The complex nature of speech signal does not permit accurate segmentation by simple or complex parameter sets adopting uniform decision schemes such as distance measures [15]. Different decision rules need to be adopted for each class. Some of these rules may be based on the behavior of adjacent segments. It is generally difficult to evolve these rules for each class, using even complex parametric representation it it has no direct relation to the waveform. Moreover certain segments are identified mainly on the basis of the parameters of the adjacent frames, which is consistent with spectrogram reading [16].

These considerations led to the choice of simple time domain parameters called ZAPDASH for segmentation in the Harpy continuous speech recognition system. Several decision rules used in the segmentation are heuristic and were evolved from experimentation with several segmentors. In this section the extraction of the ZAPDASH parameters is described. The various decision rules and the basis of some of those rules are described in Section III and IV. The performance of the segmentation procedure is compared with hand segmentation waveform in Section V and the accuracy is estimated on the basis of boundary locations. Notice that the objective of segmentation is not to determine accurate boundaries of different classes of sounds. The purpose is only to determine segments having different acoustic characteristics and place a boundary where the characteristics are likely to have changed placing more boundaries does not affect the system's recognition accuracy. In this respect this segmentation procedure is to be distinguished from those normally used in speech systems.

2.2. ZAPDASH Parameters

These are Zero-crossing And Peak to peak amplitude of Differenced And SmootHed data. The peak to peak amplitudes give information about the level of the signal. The zero-crossings give information about the significant frequency component. Thus these parameters reflect time and frequency domain characteristics in a broad sense. Values of these parameters in the low and high frequency regions are obtained from smoothed and differenced data respectively. It appears logical that even simple parameters like these should describe the gross acoustic features of a segment sufficiently well. Any complex parametric representations, although characterizing a segment more accurately, is subjected to large variations making its use more difficult for segmentation into approximately uniform acoustic segments.

The analog speech signal from a close speaking microphone is prefiltered (85-4500Hz) and sampled at 10K Hz. The samples are connected to 9 bit numbers.

Smoothing is essentially a low pass filtering operation on speech samples. In order to perform this efficiently (by shifting and addition only), a finite impulse response filter with the following coefficients was chosen: -1, 0, 1, 2, 4, 4, 4, 2, 1, 0, -1. The frequency response of the filter is shown in Figure 1. It gives an approximate low pass response with a cut-off around 1000 Hz. The smoothed output is computed only once every five sampling instants since the effect of high frequencies is negligible. This down sampling speeds up the computation of the parameters. Differencing is accomplished by merely subtracting the value of the previous sample from the present one. The purpose of differencing is to give high frequency emphasis to data as shown in Figure 1. Figure 2 illustrates typical smoothed and differenced waveforms.

A frame width of 10 msec was chosen for computing the ZAPDASH parameters. From the initial ten known silence frames the dc bias and the correction factors for noise are computed. The dc bias is the average of all the samples in the silence frames. The peak to peak amplitudes in each frame of the smoothed and differenced silence data are computed. The difference between the maximum and minimum values of these amplitudes over the ten frames for the smoothed data is denoted by ϵ_s and for the differenced data by ϵ_d. Zero crossings are counted for all the subsequent frames only when the sample values cross the zero band which is $\pm 0.25\ \epsilon_s$ for smoothed data and $\pm 0.25\ \epsilon_d$ for differenced data. The peak to peak amplitude for each frame is corrected by subtracting $0.5\ \epsilon_s$ for smoothed data and

0.5 ϵ_d for differenced data. The correction factors are applied for parameters computed for bias corrected speech data.

Speech data following the initial silence frames is corrected for dc bias by subtracting the bias value from each sample value before storing the data. By applying suitable thresholds (discussed in Section IV), the data following the ten frames of silence are tested to determine whether they belong to silence class or not. Having determined the first nonsilence frame, a few frames prior to it are also stored along with the speech data. We will use SZ and SP to denote the zero crossing counts and the peak to peak amplitudes of the smoothed data, obtained after applying the corrections for noise fluctuations. Similarly DZ and DP will denote the corresponding functions of the differenced data.

In order to compensate for changes in the overall level of the input speech signal due to variations in gain or level of speaking, statistics for SP and DP are collected as follows. The number of times the SP (or DP) gets the values of 0, 1, 2, . . . 511 are determined. From these histograms (Figure 3) the SP and DP levels, within which 90% of the SP and DP values lie, would give an indication of the overall level. The actual values of SP and DP for each frame are normalized by setting the 90% levels at 450.

3. NATURE OF SOME CLASSES OF SOUNDS

In this section we shall review the properties of some classes of sounds[17]-[18] which form the basis for segmentation procedure. The classes are stops and fricatives. These classes are specifically chosen because the vocal tract system and its excitation change rapidly and significantly during the production of these sounds. Moreover, it is often difficult to determine boundaries at which significant changes in the acoustic properties take place because the signal level is low and the properties are somewhat unpredictable for these two classes.

The main difference between stops and fricatives lies in the excitation. Stops are produced by the shock excitation of the vocal cavities due to the release of the pressure build-up behind a closure of the vocal tract. The frication, on the other hand, is the sound produced from the turbulence due to air flow through a narrow passage. If there is voicing (vibration of vocal folds) along with the normal excitations of stops and fricatives, the resulting sounds are called voiced stops and voiced fricatives.

An unvoiced initial stop in a word usually consists of a silence followed by a short period of aspiration followed by friction before the vowel begins. There is usually a measurable concentration of fricative energy in the regions of higher formants throughout the aspiration period, and it is difficult to decide whether at a given moment the pattern in these formants represented breathy phonation (aspiration) or modulated fricative energy. The onset of voicing due to the following vowel is generally distinguishable because of the weak concentration of energy at the frequency of the first formant during aspiration and friction. For the voiced initial stops the period of aspiration is absent, but the period of aspiration following the spike due to explosion is usually more prominent than in the case of voiceless stops. The final voiced stops are often produced with full voicing and a voiced release. The beginning of the final voiceless stops is determined by the abrupt cessation of all formants which is identified by the sudden decrease in energy. It is to be noted that voicing is not crucial to distinguish the stops /b/,/d/,/g/ from /p/,/t/,/k/. The difference in pressure release causes higher intensity bursts in /p/,/t/,/k/ compared to /b/,/d/,/g/. This accounts for the well known fact that the bursts /p/,/t/,/k/ are often followed by an aspiration, which is not present in /b/,/d/,/g/. Therefore, the acoustic correlates in the production of stops are rapid changes in the short-time spectrum preceded or followed by a fairly long period during which there is no energy in all the bands above the voicing component (>300 Hz).

Unvoiced fricatives are characterized by significant energy concentration in the frequency range beyond 3000 Hz. This is largely due to the nature of hiss excitation at the narrow constriction created in the vocal tract. Voiced fricatives, on the other hand, have two excitation components; hiss and voice source. For a given air pressure the acoustic intensity of hiss component of voiced fricatives is inherently less than that of the corresponding voiceless items.

Figure 4 shows ZAPDASH parameters for some examples of stops and fricatives. We shall discuss some general characteristics of the parameters, although there is considerable variation among several samples. The most important feature is that there is a sudden change in the values of one or the other of the parameters indicating the change from the adjacent vowel or silence segment. The parameters for /b/ show nearly uniform characteristics within the segment with generally higher values for SP than for DP. The beginning of /d/ consists of a silence followed by unvoiced fricative-like behavior with large values of DZ (>30). For /t/ the values of SP are lower than DP and DZ is usually greater than 25. An unvoiced fricative is indicated by a sudden change in SP or DP values with DZ value higher than 25. Voiced fricatives are usually similar to other voiced segments and hence can be grouped with them most of the time. Besides these characteristics, if there is a short duration (10 to 20 msec) segment within a long voiced speech, indicated by a sudden drop in values of SP or DP or both, it can be interpreted as a stop or an unvoiced fricative within the word. The presence of /h/, especially at the beginning of a word is usually detected by a sudden increase in the value of DZ, although its absolute value is small (10 to 40).

4. SEGMENTATION PROCEDURE

The procedure to segment an utterance into uniform acoustic segments using ZAPDASH parameters will be described in this section. The procedure consists of several heuristic decision rules based on the characteristics of different classes of sounds. The different classes into which an utterance is segmented are denoted by SIL (silence), FRC (unvoiced fricative, ASP (aspiration), PLS (plus), SPP (peak values of SP) and SPL (low values of SP). It is to be noted that this nomenclature is quite arbitrary and the classes may not strictly represent the true sound classes. However, as will be shown in Section V, boundaries are correctly placed whenever significant changes in the acoustic characteristics take place. This is the purpose of segmentation in the Harpy system as explained in the introduction.

The flow chart for the classification is illustrated in Figure 5. The various decision rules at each stage are denoted by D_k which will be explained later in this section. The first stage divides the frames into two broad classes, silence and nonsilence, using suitable thresholds for SP and DZ values. From the nonsilence group FRC class frames are isolated using a different set of thresholds for SP and DZ. In the next stage short segments (≤ 20 msec) are tested to determine whether they belong to ASP or FRC. This is performed mainly on the basis of correction rules (D_3) based on heuristics and from a knowledge of adjacent frames. Low level voices segments as in the voiced stop /b/ are called PLS (plus) and are separated from the unclassified segments after stage 3. At the end of stage 4 another set of correction rules (D_5), based on the knowledge gained so far, are applied to short segments (≤ 20 msec) in the entire utterance (except SIL and ASP classes) to reclassify the segments into SIL, FRC, PLS and the rest. The unclassified segments at this stage are checked for the presence of ASP by observing the beginning of each of these segments. The remaining unclassified frames are split into low (SPL) and high (SPP) amplitude classes. In the final stage all the long (≥ 60 msec) nonsilence segments in the entire utterance are further subdivided by observing for significant changes in the ZAPDASH parameters in the regions. This region splitting is necessary since the Harpy system works better for over segmentation but does not recognize a missing phone in under segmentation. This also helps in placing a boundary at the transition from one vowel to another.

In the segmentation a recursive subdivision of the utterance into smaller segments is performed. At several stages double thresholds for the parameters are used. One of the thresholds allows the detection of the presence of a particular class and the other threshold helps in determining the extent of the segment of that class. In the following we shall describe the various decision rules briefly.

D_1: Silence Detection

To be labeled as SIL (silence) a segment must have at least one frame with an SP value not greater than 7 and one frame with a DZ value not greater than 10. The segment will then be extended to include all adjacent frames whose SP values are not greater than 15 and DZ values are not greater than 20. Figure 6 shows the decision thresholds for SIL detection. Figure 7 shows the result of classification using the decision rule D_1.

D_2: Fricative Detection

An FRC segment must have at least one frame whose SP value is not greater than 20 and one frame whose DZ value is at least 45. The segment is then extended to include all adjacent frames with SP value not more than 40 and DZ value at least 25. Figure 8 illustrates the decision rule D_2. Figure 9 shows the result of application of D_2.

D_3: Correction Rules

The following rules are applied to merge some short (≤ 20 msec) segments.

1. One or two frames of unlabeled segment following a silence are renamed as ASP.

2. An unlabeled segment of one or two frames after a FRC is merged with the FRC.

Figure 10 illustrates the application of these correction rules.

D_4: Low Ampiltude Detection

To be labeled as PLS, there must be at least one frame with SP less than 25 and one frame with DP less than 25 and then the segment is extended until SP or DP value exceeds 50 (figure 11). Some examples of PLS detection are shown in Figure 12.

D_5: More Correction Rules

These are another set of fix-up rules applied to take care of short (m 20 msec) ambiguous segments.

1. One or two frames of PLS following a FRC is merged with the FRC.

2. One or two frames of PLS followed by FRC or SIL is merged with the FRC or SIL.

3. One or two frames of unlabeled segments following a PLS is merged with the

next segment.

In Figure 13 several examples are given to illustrate the application of these rules.

D_6: Asperation Detection

The first frame of each unlabeled segment and one frame on either side are checked for an isolated peak in the ratio DP/SP. If found, such the frame with the peak in DP/SP is designated as ASP. Example of ASP detection based on this rule is shown in Figure 14.

D_7: Dip Detection

The unlabeled segments are checked for the presence of significant dips in SP values. If such dips are found, the segments are subdivided into SP dip (SPL) and SP peak (SPP) regions. The presence of dips is identified by using the convex hull operation illustrated in Figure 15. First the largest SP value (frame 154 in Figure 15a) is located. From either end of the segment a contour connecting all the nondecreasing values of SP up to the peak is drawn as shown by the dotted line in Figure 15a. The largest difference between this contour and the SP curve is determined (at frame 166 in Figure 15a). If this difference is at least 70 and the SP value at the frame is no more than 70% of the value on the contour of the value on the contour of the convex hull, then the dip is considered significant. The convex hull contour is modified to include this dip as shown in Figure 15b. The remaining portions of the segment are analyzed further by the same convex hull operation until all the significant dips are isolated and included in the convex hull contour as shown in Figure 15c. A segment boundary is placed between each peak and the adjacent significant dip in the final convex hull contour in Figure 15c as follows. Calculate 150% of the SP value at the dip and the average of the SP values at the peak and dip. Place a boundary either at the point where SP values is just above the smaller of these two values or at the point of the greatest slope of the SP contour whichever is closest to the dip. If a boundary would be placed within three frames of the end of the segment, the new boundary will be moved back to assure a minimum length of three frames. In Figure 16 the result of the convex hull operation in SP dip detection is illustrated.

D_8: Region Splitting

The procedure described in this section breaks up large (≥ 60 msec) nonsilence segments into smaller segments whenever there is a significant change in the values of any one of the ZAPDASH parameters. To describe the significant change let us consider the SP values. The first frame of the segment is ignored because it may have transitional characteristics of the parameters. The n the running minimum and maximum SP values are set to the SP value of the second frame initially. Starting from the third frame onwards the minimum and maximum of SP values up to that point are computed. If the new minimum, multiplied by a factor (to be discussed), is less than the new maximum at any frame then it indicates placement of a segment boundary. The multiplication factor starts at 2.4 for the third frame. Thus it requires very large change in ZAPDASH parameters to force a short segment. However, the factor is progressively reduced by 0.1 for each additional frame tested, until the fifteenth frame and for further frames the factor is kept constant at 12.

The spread of values in any one of the ZAPDASH parameters can force the placement of a segment boundary. First the frame that has the greatest slope in the value of the significant parameter is determined. Then a boundary is placed at a point, in the vicinity (± 2 frames) of the above frame, where the majority of the ZAPDASH parameters have greatest slope. The procedure looks for such a frame only in the second half of the segment up to the present where the boundary placement was forced by the significant parameter.

Skipping one frame from the new boundary, this region splitting algorithm is executed over the remaining portion of the original segment. Figure 17 illustrates the application of the region splitting algorithm.

5. PERFORMANCE EVALUATION

In this section we shall describe the result of the segmentation procedure discussed earlier. First we shall consider the application of the procedure on one utterance in detail. The performance of the segmentation is then evaluated by comparing the final segmentation with the result of hand segmentation of the waveform of the utterance.

The utterance, "DO YOU HAVE ANY NEW PAPERS ON SPEECH UNDERSTANDING" spoken by a male speaker, is considered for illustration. All the simple values of the speech data are represented as 9 bit positive numbers i.e., the values range from 0 to 511. The length of the utterance is 235 centiseconds starting from an arbitrary time reference of 48 centiseconds and extending up to 283 centiseconds. The total number of frames is therefore 235.

The peak to peak amplitudes of the differenced (DEPS) and the smoothed (SEPS) data and the bias values for the first ten silence frames are given below.

FRAME NUMBER	DEPS	SEPS	BIAS
1	6	2	2
2	3	3	2
3	4	2	2
4	4	2	3
5	3	2	2
6	4	2	3
7	3	2	2
8	3	2	3
9	2	3	3
10	4	2	3
Average values	3	2	252

The average values are the truncated values of the actual average over the ten frames. The maximum values of SP and DP are 580 and 610 respectively. Although the speech data has values in the range 0 to 511, the smoothed and differenced speech data can have values in a much higher range than 512. The 90% values of SP and DP are 343 and 378 respectively which are obtained from the histograms for the values of SP and DP. Using the average values for the bias, DEPS, and SEPS the values of the corrected ZAPDASH parameters for all the frames are computed and stored.

When the decision rule (D_1) for silence detection is applied the following segmentation is obtained. The numbers here refer to time in centiseconds and NIL refers to the unlabeled frames.

Segmentation at the first stage 1:

```
 48  SIL   61  NIL  126  SIL  130  NIL  146  SIL  150  NIL
183  SIL  188  NIL  199  SIL  201  NIL  282
```

For all the unlabeled segments above, the decision rule (D_2) for FRC detection is used to split them into FRC and NIL where NIL refers to the unlabeled segments at the end of stage 2:

```
 61  FRC   64  NIL  126
130  NIL  131  FRC  133  NIL  146
150  NIL  159  FRC  163  NIL  176  FRC  182  NIL  183
188  NIL  199
201  FRC  208  NIL  229  FRC  241  NIL  282
```

At this stage the correction rules D_3 determine that the frame 130-131 is designated as ASP and that the frame 182-183 is merged with the FRC in 176-182.

The unlabeled segments (NIL) are checked for the presence of PLS using the rules D_4. Segmentation at the fourth stage:

```
 64  NIL  126
133  NIL  145  PLS  146
150  NIL  159
163  NIL  176
188  NIL  198  PLS  199
208  NIL  229
241  NIL  272  PLS  282
```

The correction rules D_5 combine the frame 145 PLS 146 with the segment 146 SIL 150 and the frame 198 PLS 199 with the segment 199 SIL 201.

The rule D_6 uses the ratio DP/SP and determines that the segments 150-151 and 188-189 belong to ASP class.

Results of SP dip detection from convex hull operation are summarized below.

```
 64  SPP  [Max Slope @ 85;  Thr @ 87;  1.5 SPD = 184, Av = 441]
 87  SPL  [Max Slope @ 90;  Thr @ 90;  1.5 SPD = 184, Av = 356]
 90  SPP  [Max Slope @ 99;  Thr @ 99;  1.5 SPD = 228, Av = 370]
 99  SPL  [Max Slope @ 101; Thr @ 101; 1.5 SPD = 228, Av = 249]
101  SPP  [Max Slope @ 109; Thr @ 109; 1.5 SPD = 153, Av = 224]
109  SPL  [Max Slope @ 116; Thr @ 116; 1.5 SPD = 153, Av = 188]
116  SPP   126
133  SPP   145
151  SPP   159
163  SPP   176
189  SPP   198
208  SPP  [Max Slope @ 212; Thr @ 215; 1.5 SPD = 102, Av = 216]
215  SPL  [Max Slope @ 223; Thr @ 223; 1.5 SPD = 102, Av = 107]
223  SPP   229
241  SPP  [Max Slope @ 246; Thr @ 261; 1.5 SPD = 54, Av = 216]
261  SPL  [Max Slope @ 265; Thr @ 265; 1.5 SPD = 54, Av =  68]
265  SPP   272
```

The parameters in the brackets are computed to determine the boundary. For example, in the interval between the first peak and the following dip in the convex hull contour for SP, the maximum slope occurs at 85 and the threshold at 87. The threshold is determined by the minimum of 1.5 times the SP dip (SPD) value and the average of the peak and the dip values. In this case the minimum of 184 and 441 is 184 and it occurs at 87. Since 87 is closer to the dip, a boundary is placed at 87. A similar procedure, to determine the other boundary for SPL, is followed in the interval between the first dip and the next peak. The boundary appears at 90 and therefore the interval 64 to 87 is designated as SPP and the interval 87 to 90 as SPL. The procedure is continued along the convex hull contour to determine all SPL segments.

Segmentation at the end of Stage 7:

BEGIN TIME	SEGMENT CLASS	END TIME	DURATION (c sec)
48	SIL	61	13
61	FRC	64	3
64	SPP	87	23
87	SPL	90	3
90	SPP	99	9
99	SPL	101	2
101	SPP	109	8
109	SPL	116	7
116	SPP	126	10
126	SIL	130	4
130	ASP	131	1
131	FRC	133	2
133	SPP	145	12
145	SIL	150	5
150	ASP	151	1
151	SPP	159	8
159	FRC	163	4
163	SPP	176	13
176	FRC	183	7
183	SIL	188	5
188	ASP	189	1
189	SPP	198	9
198	SIL	201	3
201	FRC	208	7
208	SPP	215	7
215	SPL	223	8
223	SPP	229	6
229	FRC	241	12
241	SPP	261	20
261	SPL	265	4
265	SPP	272	7
272	PLS	282	10

The final stage of segmentation consists of region splitting, the results of which are summarized below.

Summary of region splitting operation:

```
[MF = 2.0] DP (222,250;444) DZ (42,46,84) SP (157,367;314) SZ (5,6;10)
break caused by SP at 71. max slope at 71 (0 0 1 2 1) seg at 72

[MF = 2.0] DP(400,652:800) DZ(12,27:24) SP(400,750:840) SZ(9,12:18)
break caused by DZ at 79. max slope at 77 (0 0 4 0 0) seg at 77

[MF = 2.0] DP(323,685:646) DZ(12,33:25) SP(400,760:840) SZ(8,12:18)
break caused by DP at 84. max slope at 84 (0 0 1 1 2) seg at 86
moving seg back to 84

[MF = 1.9] DP(38,171:72) DZ(6,38:52) SP(85,275:384) SZ(3,7:12)
break caused by DP at 124. max slope at 124 (0 0 4 0 0) seg at 124
moving seg back to 123

[MF = 1.7] DP(354,614:601) DZ(33,43:59) SP(300,498:540) SZ(3,9:9)
break caused by DP at 143. max slope at 140 (0 1 2 0 1) seg at 140

[MF = 2.0] DP(177,373:354) DZ(24,28:50) SP(288,451:667) SZ(9,11:18)
break caused by DP at 170. max slope at 170 (0 0 1 2 1) seg at 171

[MF = 2.2] DP(73,134:160) DZ(47,70:103) SP(10,35:22) SZ(2,9:11)
break caused by SP at 181. max slope at 181 (0 0 2 2 0) seg at 181
moving seg back to 180

[MF = 2.2] DP(16,40:35) DZ(5,9:11) SP(68,94:200) SZ(4,5:11)
break caused by DP at 220. max slope at 220 (0 0 2 1 1) seg at 220

[MF = 1.9] DP(52,173:98) DZ(60,67:120) SP(2,9:20) SZ(0,5:10)
break caused by DP at 237. max slope at 237 (0 0 2 1 0) seg at 237

[MF = 2.0] DP(147,321:294) DZ(39,45:81) SP(163,288:384) SZ(7,10:14)
break caused by DP at 248. max slope at 246 (0 0 3 1 0) seg at 246

[MF = 1.6] DP(30,191:48) DZ(22,46:57) SP(66,190:265) SZ(4,9:10)
break caused by DP at 257. max slope at 257 (0 1 3 0 0) seg at 257

[MF = 2.0] DP(4,8:20) DZ(3,14:10) SP(18,26:42) SZ(2,4:10)
break caused by DZ at 279. max slope at 279 (0 1 2 1 0) seg at 279
```

In the first segment 64-87 the first significant change is detected by SP at 71. The running minimum (157) multiplied by the factor MF (2.0 in this case) yields 314 which is less than the running maximum (367). It is to be noted that for all other parameters the running minimum multiplied by MF gives the third entry in the parenthesis, which is greater than the running maximum (second entry). The maximum slope in the significant parameter (SP in this case) has occured at 71. Therefore in the range 69-73 (71 ± 2) the frames at which the greatest slopes in the ZAPDASH parameters occur are determined. The values (0 0 1 2 1) indicate that no parameters have their greatest slope at the frames 69 and 70, and one frame has its greatest slope at 71, two at 72 and one at 73. The boundary is therefore placed at 72. The segment from 72 onwards is considered again for further splitting. The final segmentation result is shown in Figure 18. A listing of the program for segmentation in SAIL language is given in the Appendix.

The performance of the segmentation is evaluated by comparing the results with hand segmentation of the waveforms for several utterances. A total of 34 utterances was considered. The waveforms of these utterances were segmented by observation. It is not difficult to determine from the waveform the boundaries at which significant changes take place except in the intervals of long vowels or vowel transitions as in diphthongs. While comparing with the machine segmentation, errors up to ±1 frames are generally ignored. This is because some boundaries cannot be determined very accurately especiallly when a transition occurs within a frame. Only the presence of the boundaries of hand segmentation is checked in the machine segmentation. If the expected boundaries is not found, then an error is marked. The extra boundaries in the machine segmentation are ignored for purposes of comparison. This is because the scheme is basically designed to provide extra boundaries for improving the recognition accuracy of the Harpy system. The table below summarizes the performance characteristics.

Summary of segmentation performance:

Number of M/C segments	Number of Hand segments	Number of extra boundaries	Number of errors
1214	947	267	37

The nature of errors is illustrated in the example shown in figures 19 and 20. The thick vertical lines correspond to hand segmentation of the waveform. It is to be noted that the errors caused by machine segmentation in figure 19 are at locations in voiced positions where the changes in amplitude do not necessarily mean changes in the acoustic characteristics. Most of the errors are of this kind although there are cases as shown in figure 20 where the missing boundary in machine segmentation at 112 csec may be critical. A careful analysis shows that out of the 37 errors there may be less than ten critical errors which are due to missing boundaries in the machine segmentation. Moreover, in most cases there is usually a boundary within ± 2 frames from the expected one in the machine segmentation. Hence the errors caused by the segmentation procedure do not affect the performance of the recognition system significantly. The total number of frames in the 34 utterances was 5762.* Since there are 1214 machine segments, the average duration of a segment is 4.74 csec. The average duration of a segment is an indication of the efficiency of the segmentation procedure, the longer the duration the better the procedure. This is because time for labeling and the following search procedure for the recognition are dependent directly on the total number of segments in the utterance.

6. CONCLUSIONS

We have described a segmentation procedure based on simple time domain parameters and heuristic decision rules. The procedure yields segmentation sufficiently accurate for obtaining good recognition in the Harpy System. It is to be noted that the purpose of the segmentor is not to determine phonemic boundaries but to break the utterance into nearly uniform acoustic segments. The procedure is designed to yield sufficiently large numbers of segments for good performance of the Harpy system. Placing more boundaries does not affect the recognition accuracy but slows down the recognition process. In this respect this segmentation scheme is to be distinguished from those normally used in speech recognition studies.

It is quite likely that the segmentation procedure may not be robust enough to work equally well for input speech corrupted with noise and distortion. This may be partly due to the decision rules which are based on the parameter values for clean speech. Our future research is directed towards the issues of noise and distortion and their effect on segmentation.

7. REFERENCES

[1] D. R. Reddy, "Speech Recognition by machine: A review", Proc. IEEE, vol 64, pp. 501-531, 1976

[2] N. Chomsky and M. Halle, The Sound Pattern of English, Harper and Row, publishers, New York, 1968

[3] R. Schwartz, and J. Makhoul, "Where the phonemes are: Dealing with ambiguity in acoustic-phonetic recognition",IEEE Trans. Acoustic, Speech, and Signal Processing, vol Assp-23, pp. 50-53, 1975

[4] B. Yegnanarayana, "Effect of noise and distortion in speech on parametric extraction", Proc. IEEE Conf. Acoustics, Speech, and Signal Processing, Philadelphia, Pa., pp. 336-339, 1976

[5] B. T. Lowerre, "The HARPY Speech Recognition System", PhD thesis, Computer Science Department,Carnegie-Mellon University, Pittsburgh, Pa, 1976

[6] C. G. Fant and B. Londblom, "Studies of minimal speech aound units", Quart. Progr. Status Rep., Speech Transmission Labs., KTH, Stockholmn, Sweden, pp. 1-11, Apr.-June 1961

[7] D. R. Reddy, "Segmentation of Speech Sounds", J. Acoust. Soc. Amer., vol 40, pp. 307-312, 1966

[8] D. R. Reddy,and P.J. Vicens, "A Procedure for Segmentation of Connected Speech", J. Audio Eng. Soc., vol 16, pp. 404-412, 1968

[9] C. C. Tappert, N. R. Dixon, A. S. Rabinowitz, and W. D. Chapman, "Automatic Recognition of continuous speech utilizing dynamic segmentation, dual classification, sequential decoding and error recovery", Rome Air Development Center, Griffiss AFB, Rome, NY, Tech. Rep. TR-71-146, 1971

[10] N. R. Dixon and H. F. Silverman, "A description of a parametrically controlled modular structure for speech processing", IEEE Trans. Acoustics, Speech and Signal Processing, vol Assp-23, pp. 87-91, Feb 1975

[11] N. R. Dixon and H. F. Silverman, "A general language-operated decision implementation (GLODIS): Its application to continuous speech segmentation", IEEE Trans. Acoustics, Speech and Signal Processing, vol Assp-24, pp. 137-162, 1976

[12] J. M. Baker, "A new time-domain analysis of human speech and other complex waveforms", PhD thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pa., 1975

[13] H. G. Goldberg, "Segmentation and labelling of speech: A comparative performance evaluation", PhD thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pa., 1975

[14] C. J. Weinstein, S. S. McCandless, L, F. Mondehein, and R. W. Zue, "A system for acoustic-phonetic analysis of continuous speech",IEEE Trans. Acoustics, Speech and

Signal Processing, vol Assp-23, pp. 54-67, 1975

[15] B. S. Atal and L.R. Rabiner, "A pattern recognition approach to voiced-unvoiced-silence classification with applications to speech recognition", IEEE Trans. Acoustics, Speech and Signal Processing, vol Assp-24, pp. 201-212, 1976

[16] D. H. Klatt and K. N. Stevens, " On the automatic recognition of continuous speech: Implications from a spectrogroam-reading experiment", IEEE Trans. Audio Eletroacoustics, vol AU-21, pp. 210-217, 1973

[17] P. Stevens, "Spectra of fricative noise in human speech", Language and Speech, vol 3, pp. 32-49, 1960

[18] M. Halle, G. W. Hughes and J. P. A. Radley, "Acoustic properties of stop constants", J. Acoustical soc. Amer., vol 29, pp. 107-116, 1957

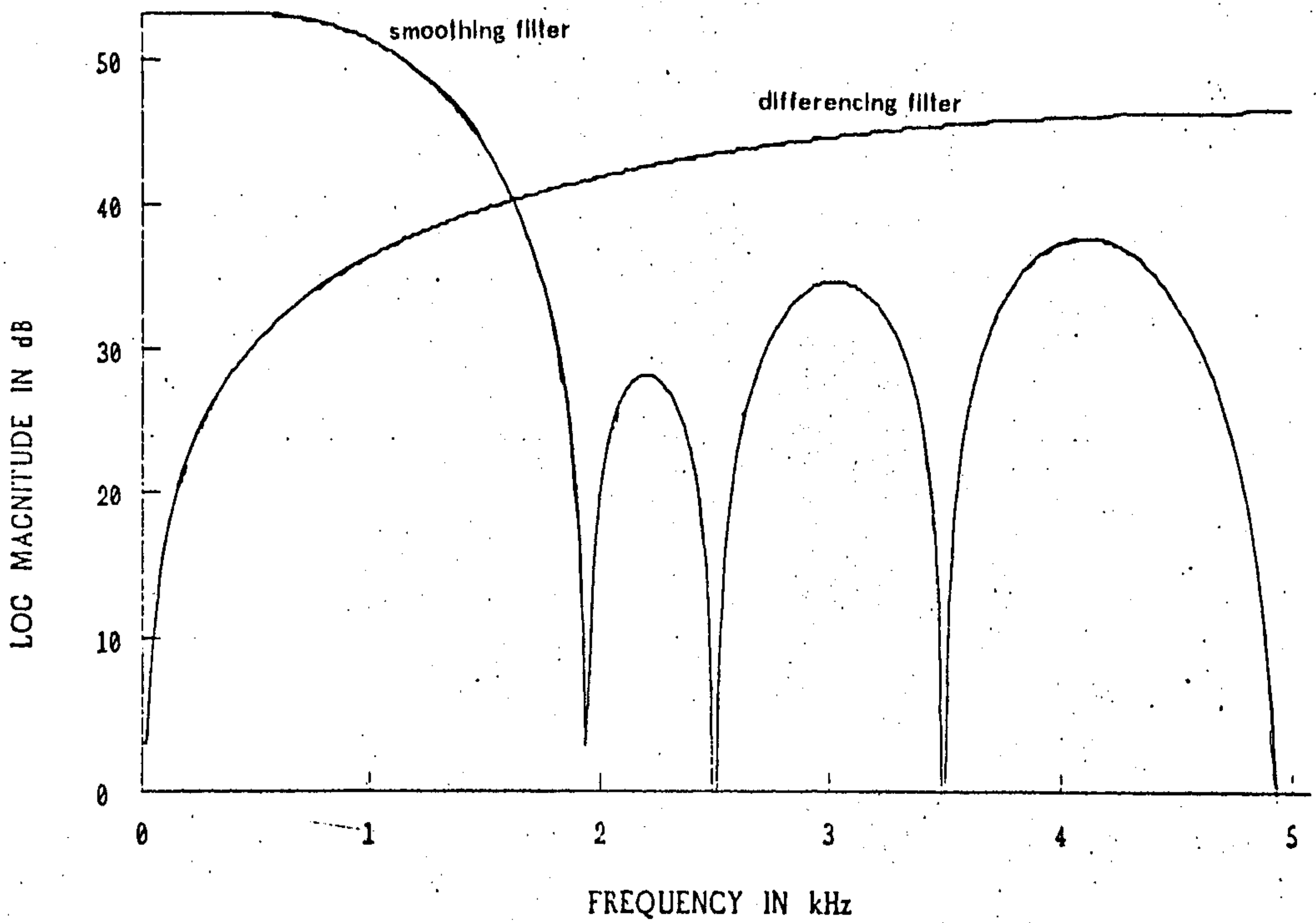

Figure 1: Frequency response of smoothing and differencing filters

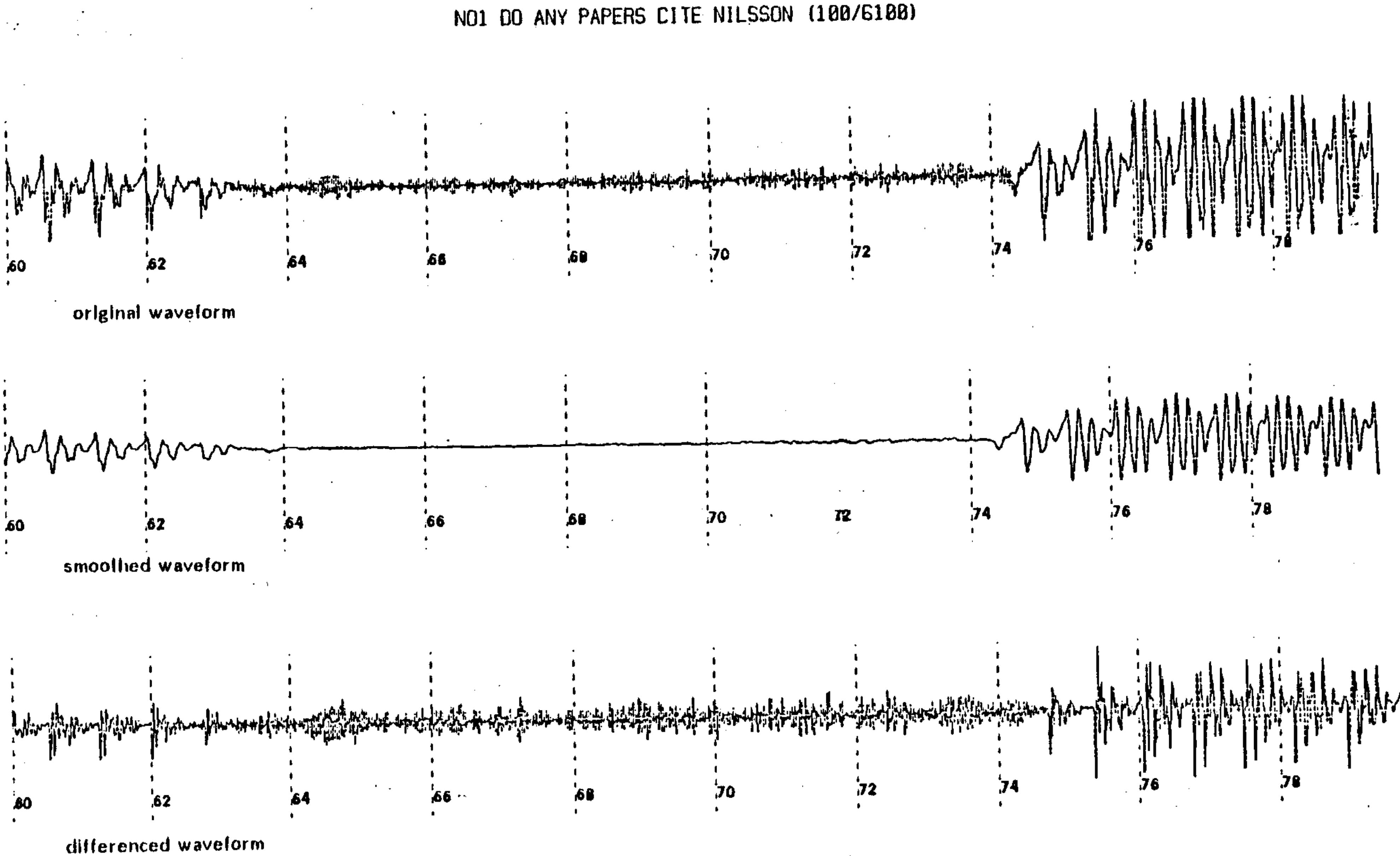

Figure 2: Example of smoothed and differenced waveforms

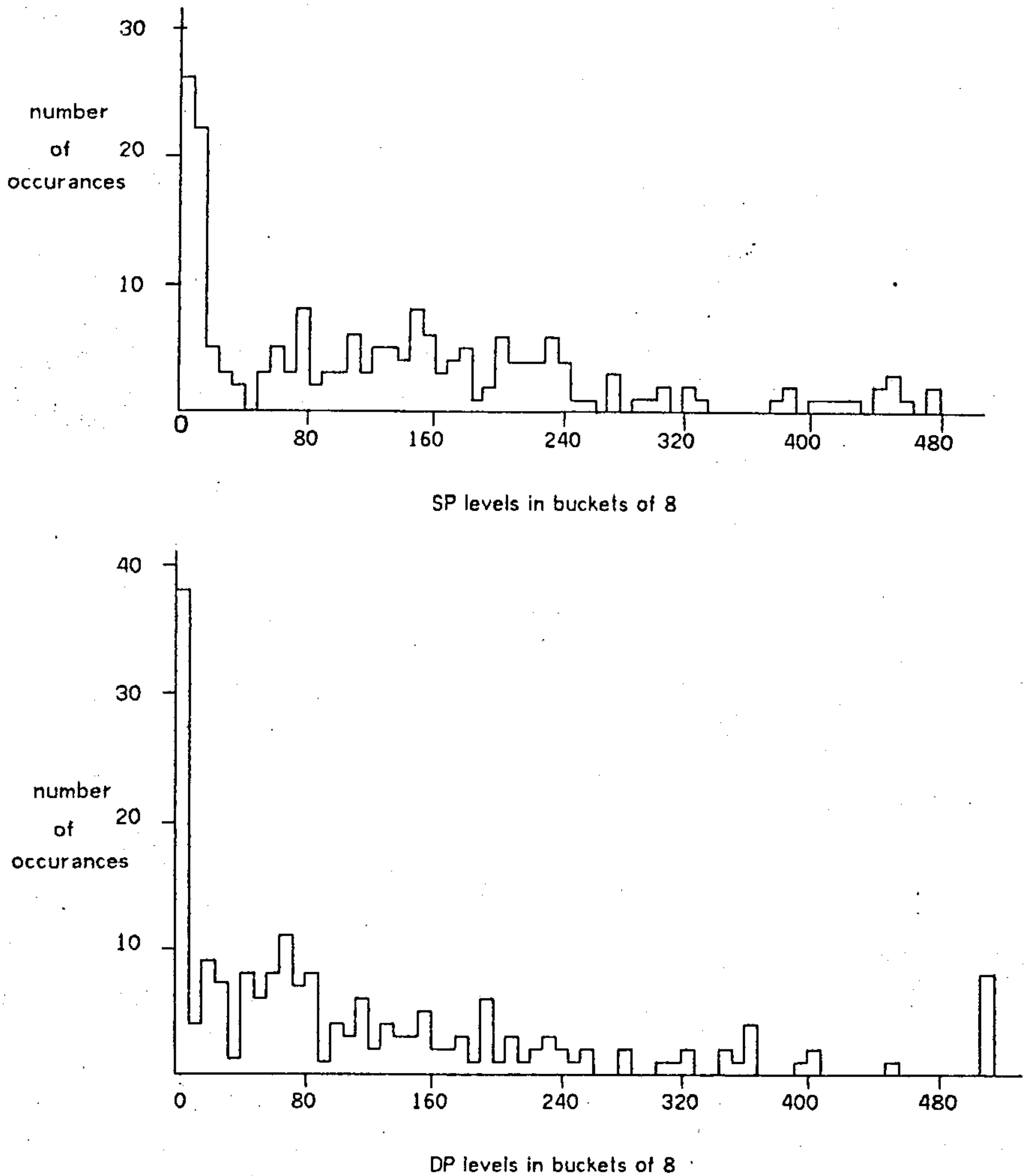

Figure 3: Histogrammes of SP and DP levels

LAC12: IS RESOLUTION THEOREM PROVING MENTIONED IN AN ABSTRACT

c sec	DP	DZ	SP	SZ	
301:	581	19	567	12	a
302:	312	16	452	9	a
303:	7	10	36	4	b
304:	7	5	26	5	b
305:	3	2	21	2	b
306:	3	3	23	2	b
307:	0	0	21	3	b
308:	5	2	17	2	b
309:	63	51	13	7	s
310:	113	64	10	4	s

LAD3: DO YOU HAVE ANY NEW PAPERS ON SPEECH UNDERSTANDING

c sec	DP	DZ	SP	SZ	
59:	1	0	2	0	-
60:	2	2	1	0	-
61:	46	36	10	4	d
62:	78	64	10	1	d
63:	78	45	23	3	d
64:	55	35	116	4	uw
65:	136	53	132	5	uw
66:	229	44	157	5	uw

LAD4: GIVE ME THE DATE OF THAT ABSTRACT

c sec	DP	DZ	SP	SZ	
197:	174	16	173	8	a
198:	101	28	108	8	a
199:	55	11	23	2	a
200:	1	0	8	1	-
201:	0	0	1	0	-
202:	1	0	1	0	-
203:	2	1	2	1	-
204:	2	2	2	1	-
205:	2	3	1	0	-
206:	2	1	3	1	-
207:	8	3	6	3	-
208:	6	0	6	2	-
209:	0	0	1	0	-
210:	0	0	1	0	-
211:	51	14	14	5	t
212:	74	64	14	7	t
213:	56	46	10	3	t
214:	25	46	6	8	t
215:	14	34	4	5	t
216:	10	21	5	5	t
217:	4	11	3	3	t

Figure 4: Examples of ZAPDASH parameters for /b/ /d/ /t/

LAD3: DO YOU HAVE ANY NEW PAPERS ON SPEECH UNDERSTANDING

c sec	DP	DZ	SP	SZ	
226:	96	38	123	5	r
227:	97	38	102	4	r
228:	104	57	48	3	r
229:	155	56	11	2	s
230:	166	64	11	4	s
231:	115	65	7	1	s
232:	120	67	7	1	s
233:	173	65	9	5	s
234:	148	60	9	3	s
235:	102	67	9	0	s
236:	103	64	6	0	s
237:	52	62	2	0	s
238:	30	66	2	0	t
239:	57	63	6	0	t
240:	53	59	7	0	t
241:	45	52	53	1	a
242:	195	37	182	7	a
243:	259	40	254	8	a

LAD2: HAVE ANY NEW PAPERS BY NEWELL APPEARED

c sec	DP	DZ	SP	SZ	
59:	1	0	1	2	-
60:	3	5	1	0	-
61:	9	29	4	8	h
62:	22	32	14	3	h
63:	32	38	18	9	h
64:	83	33	115	7	a
65:	192	25	219	8	a
66:	251	27	334	9	a

Figure 4: Examples of ZAPDASH parameters for |s| |h|

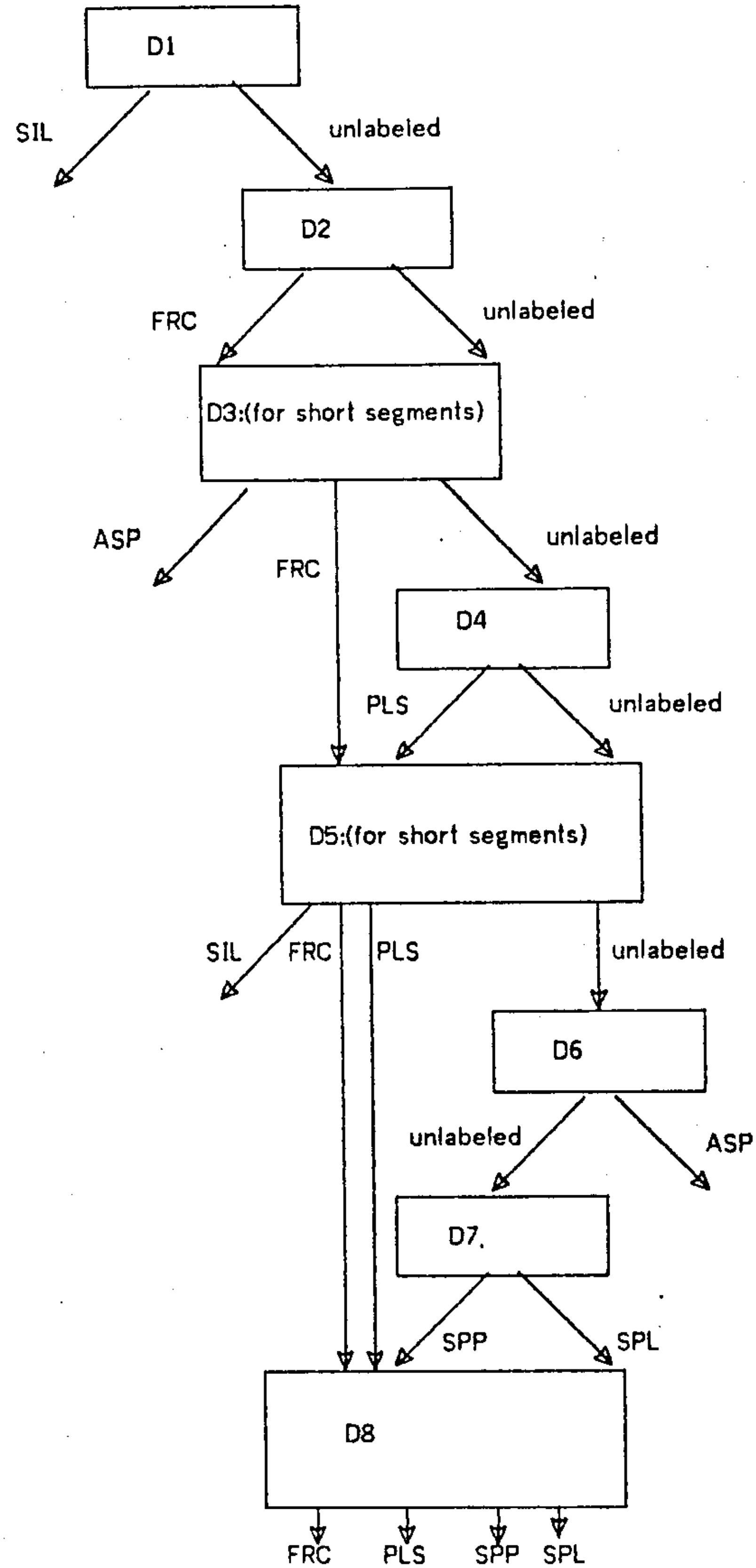

Figure 5: Flow chart for clasification of segments

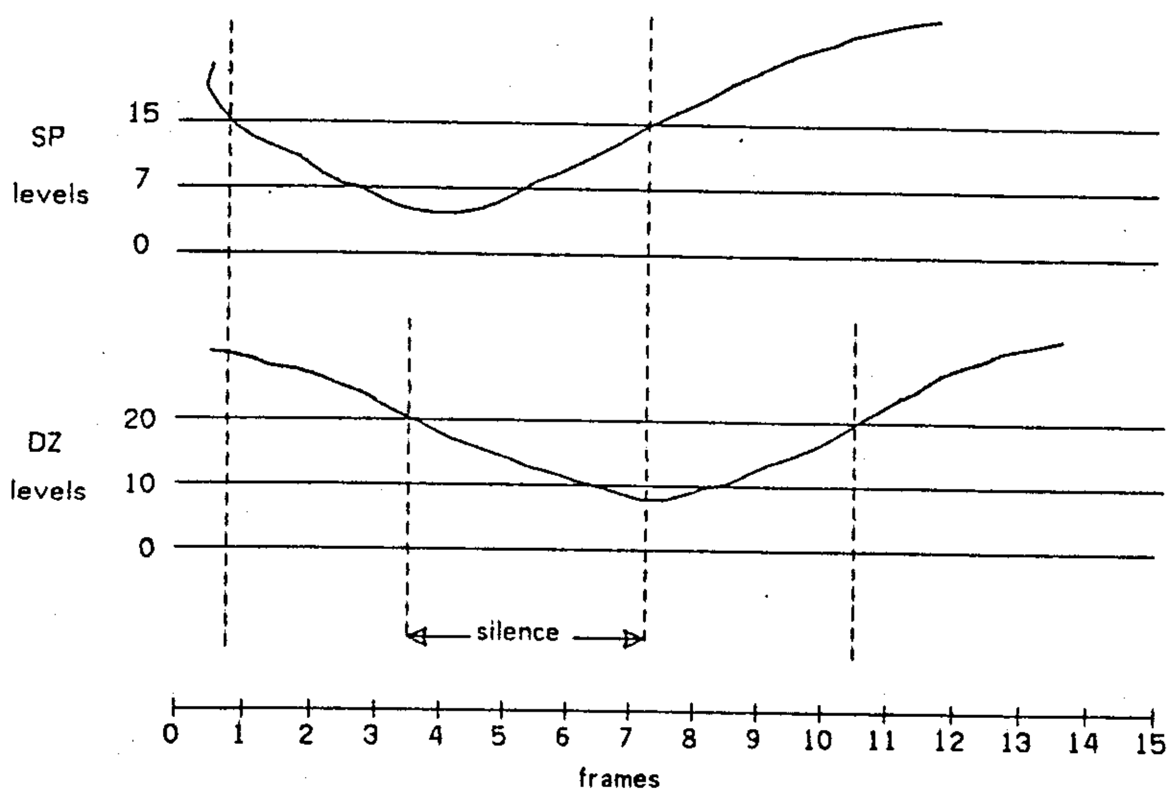

Figure 6: Silence detection by double thresholds

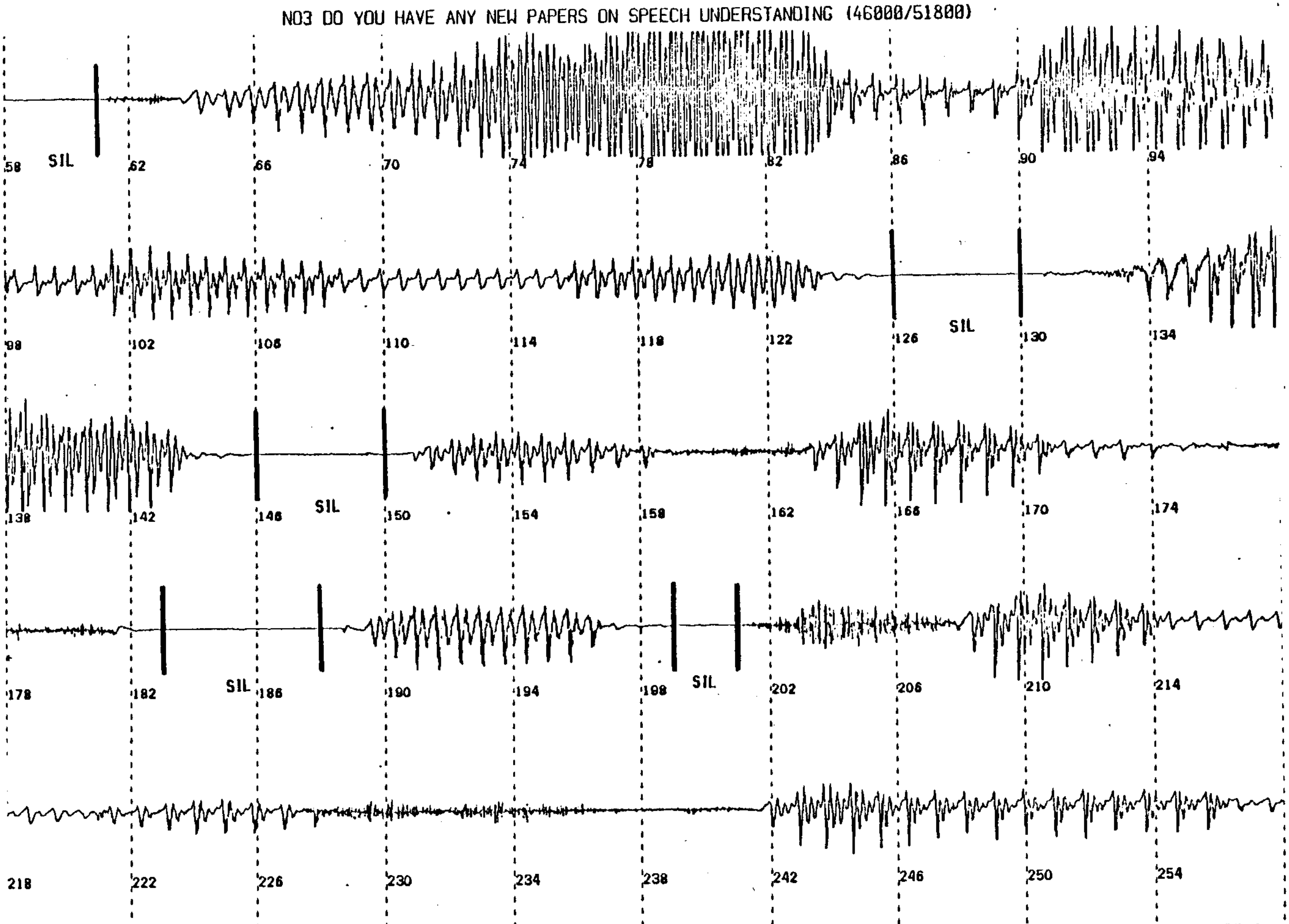

Figure 7: Result of application of decision rule D1

LAD.ADC LAD.UTT 30-May-78 12:49 UTT #: 3 PAGE 1 OF 2

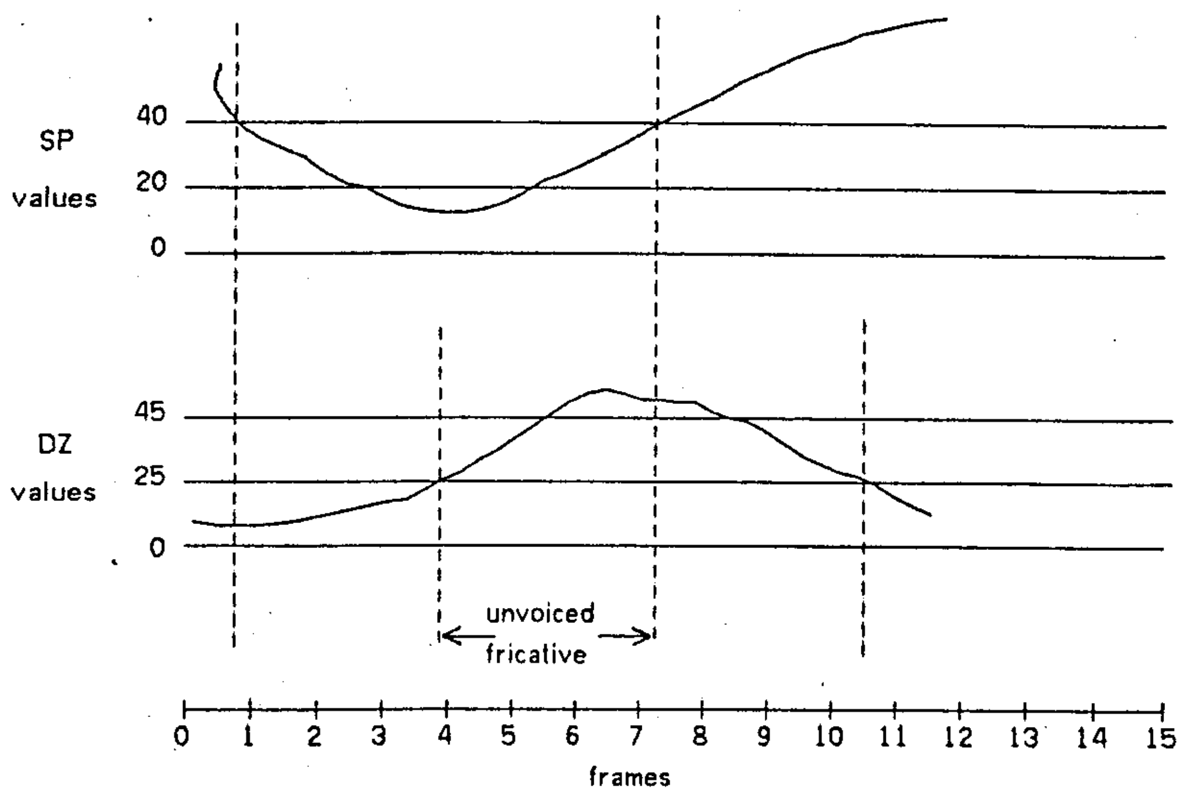

Figure 8: Unvoiced fricative detection

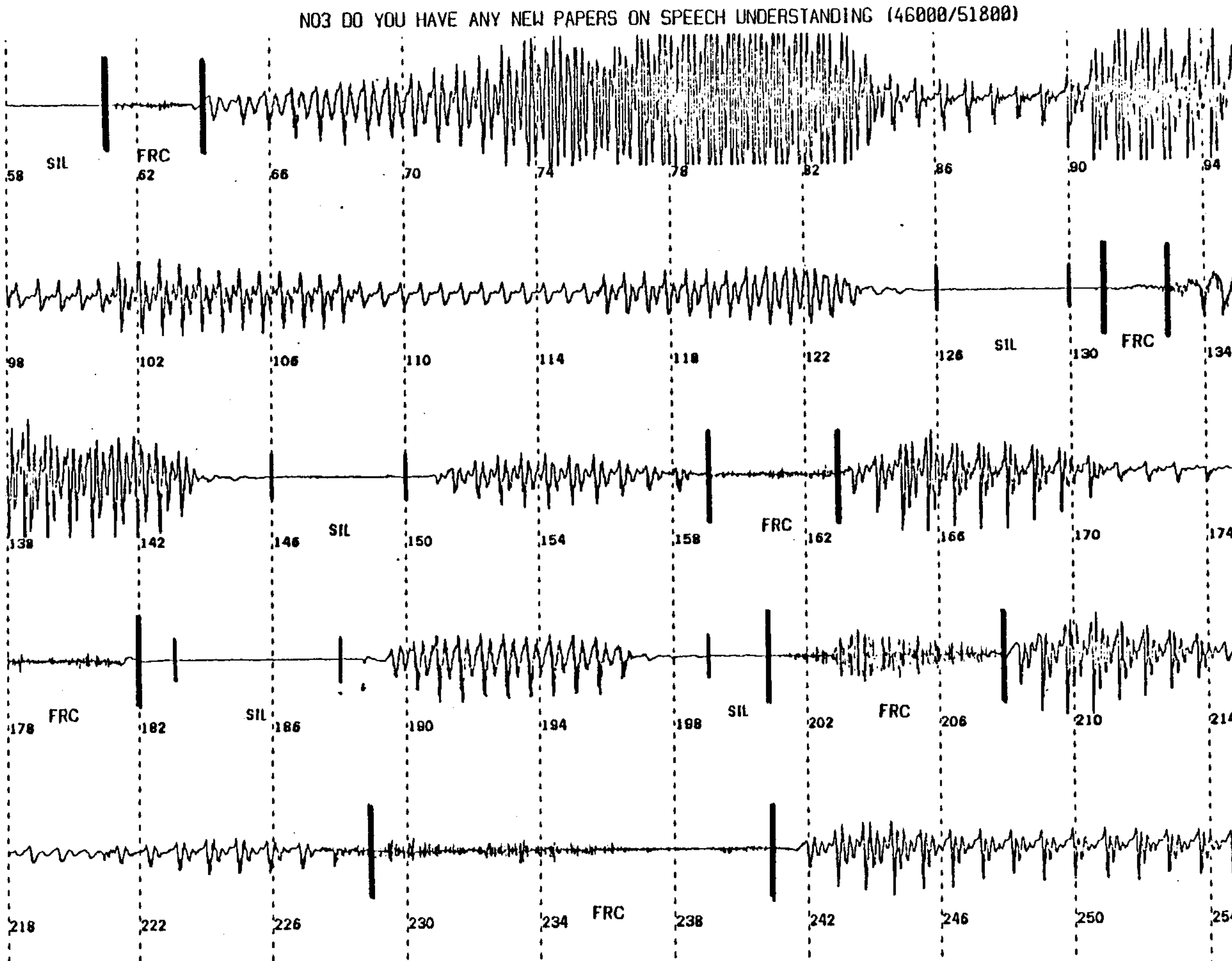

Figure 9: Result of application of decision rule D2

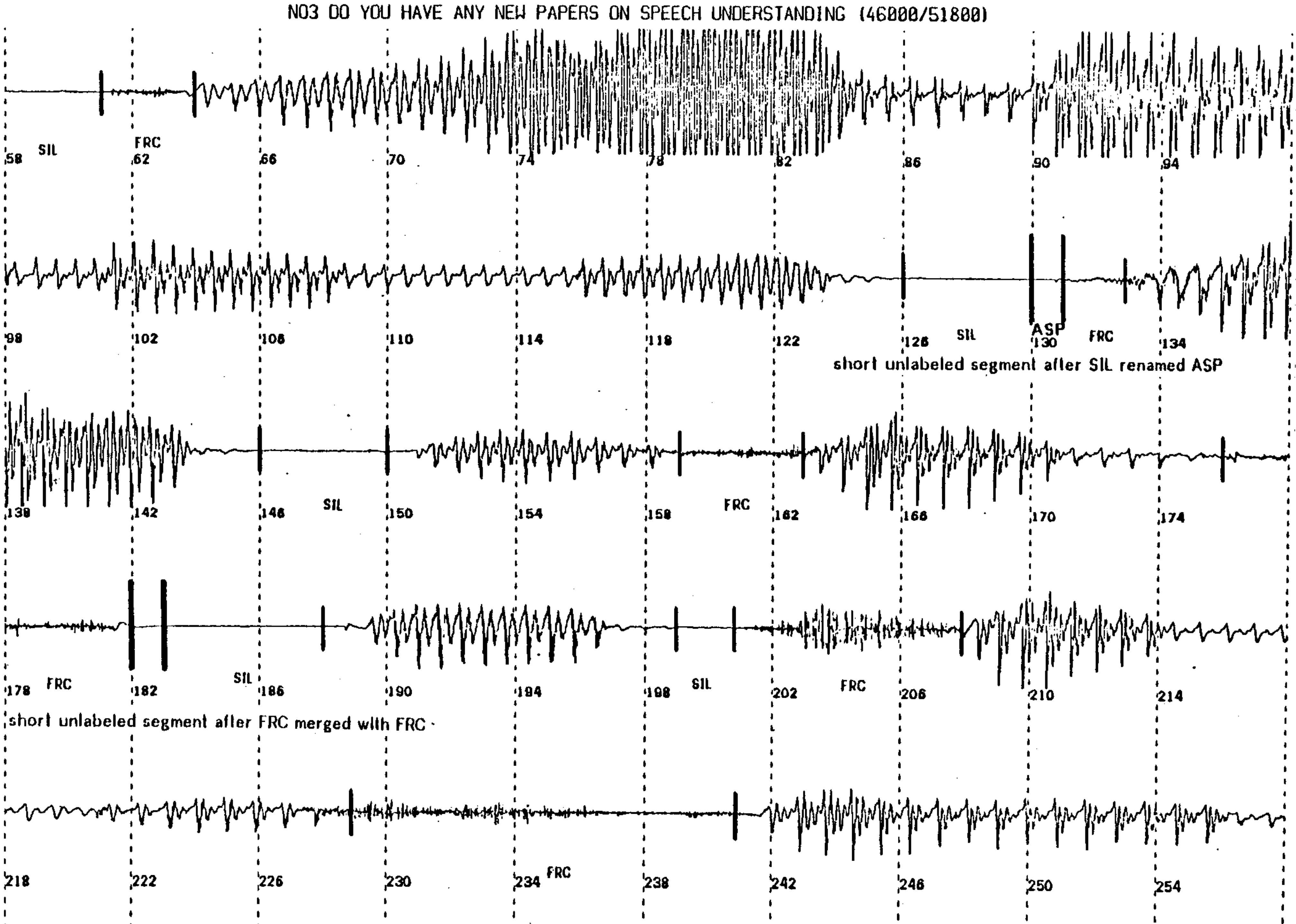

Figure 10: Eamples of correction rules D3

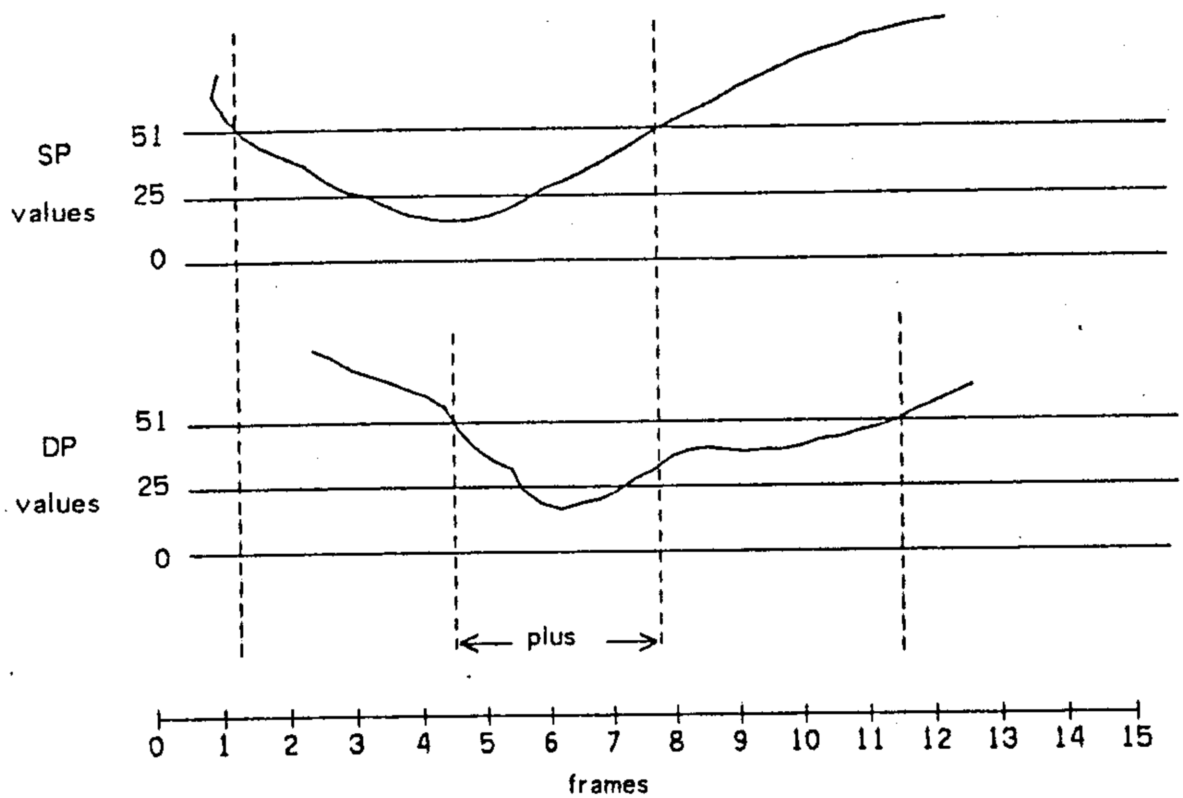

Figure 11: Plus detection

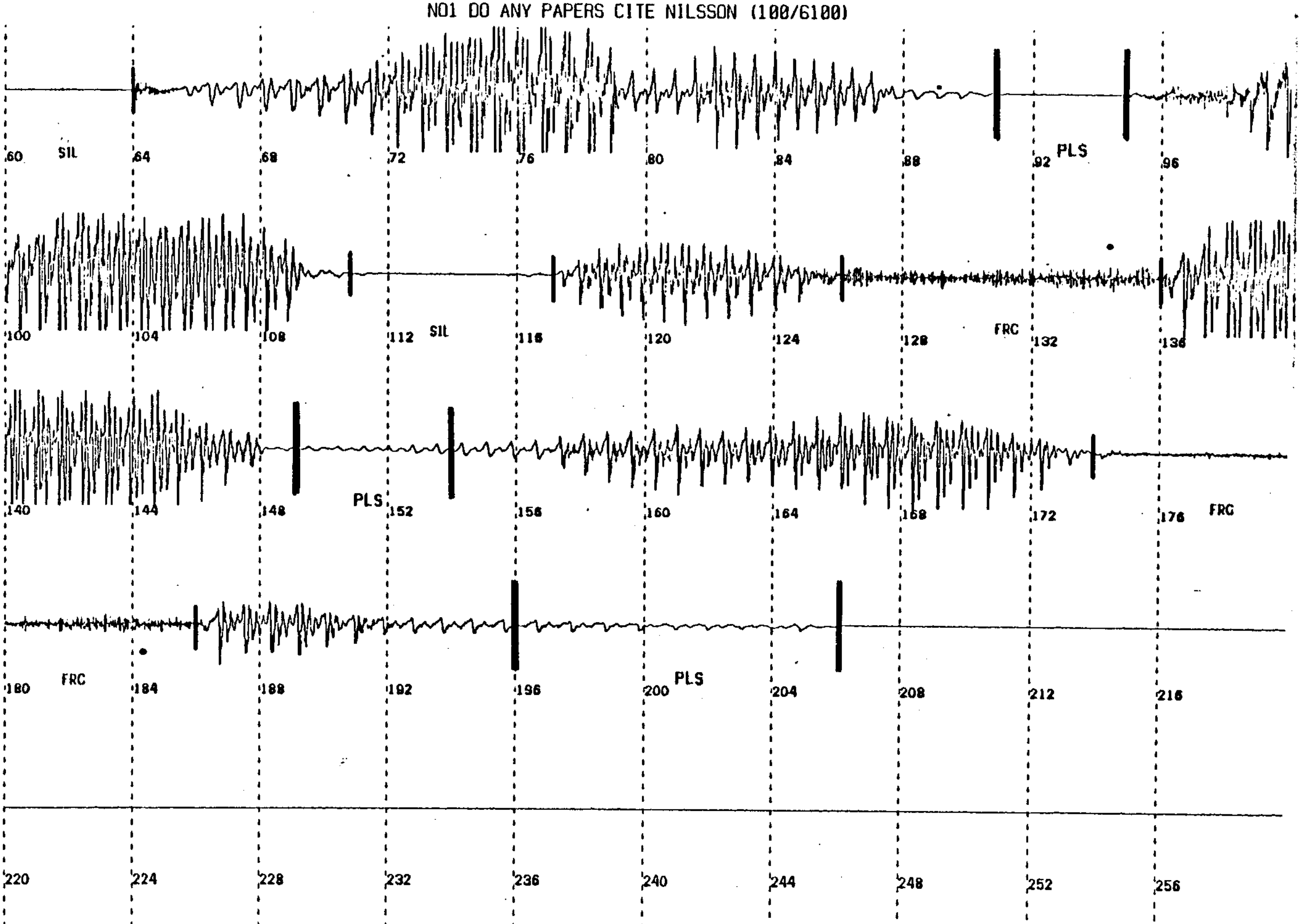

Figure 12: Result of application of decision rule D4

LAD.ADC LAD.UTT 30-May-78 12:49 UTT #: 1 PAGE 1 OF 1

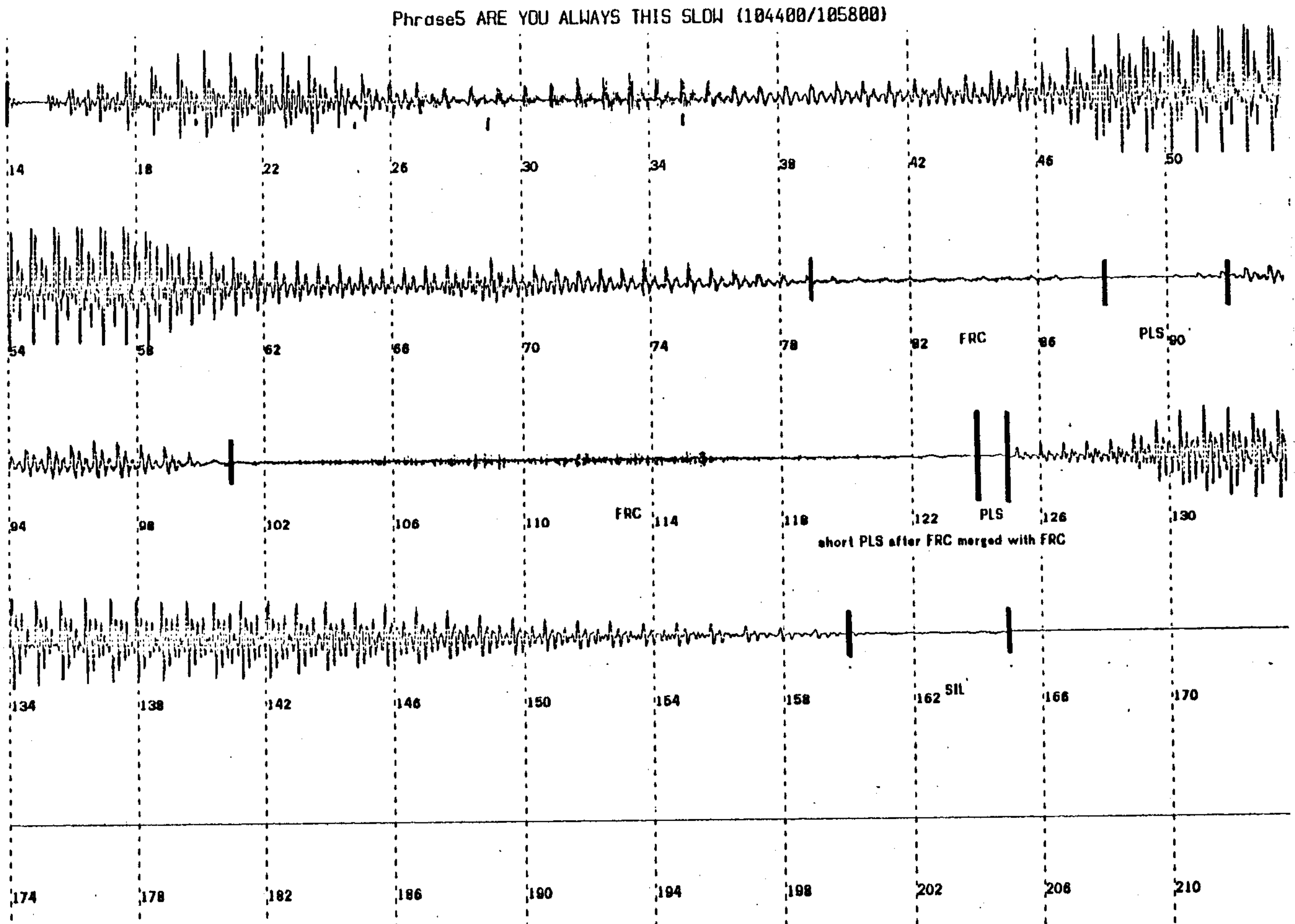

Figure 13: Examples of correction rules D5

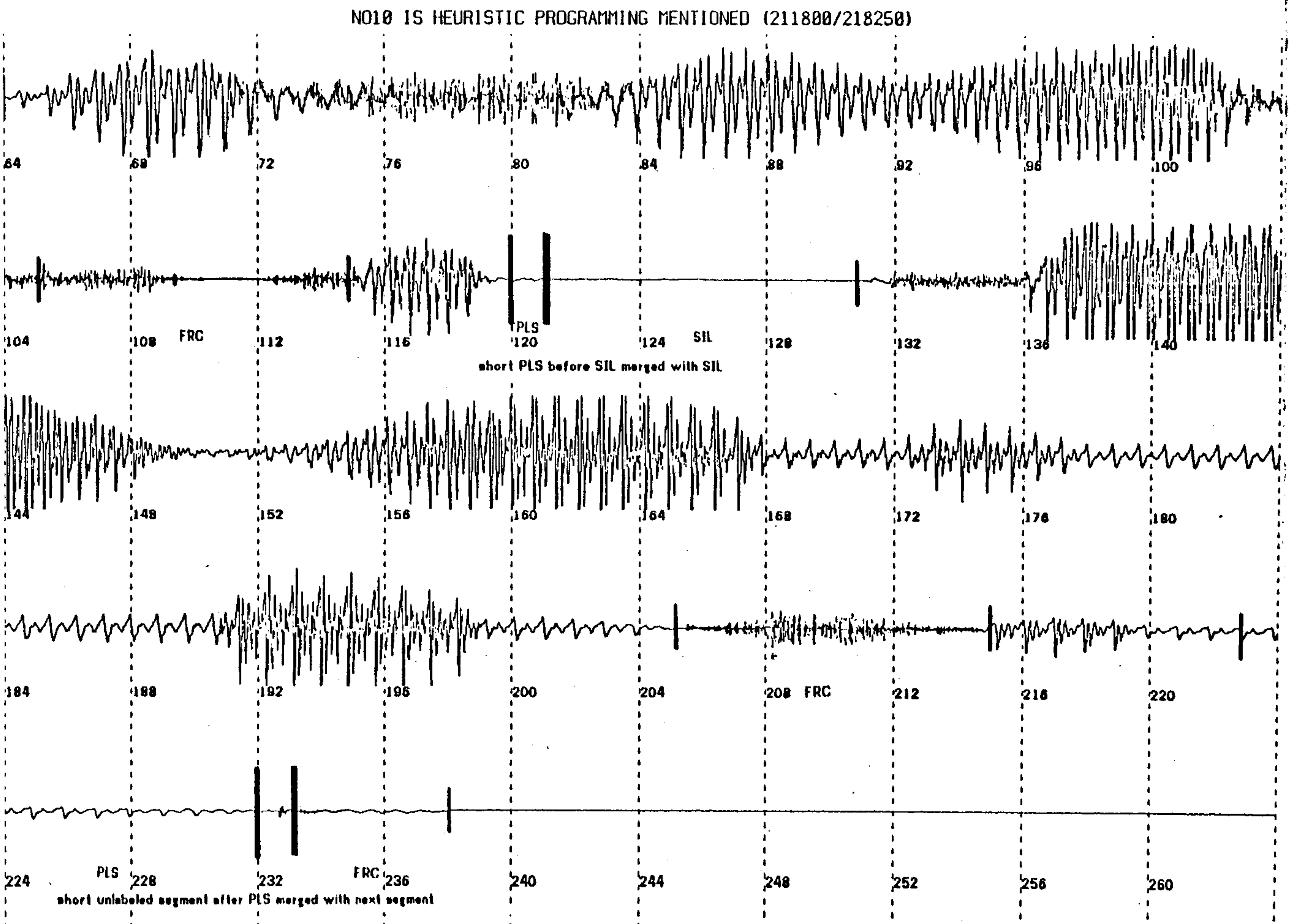

Figure 13: Examples of correction rules D5

LAD.ADC LAD.UTT 30-May-78 12:49 UTT #: 10 PAGE 1 OF 1

NO3 DO YOU HAVE ANY NEW PAPERS ON SPEECH UNDERSTANDING (46000/51800)

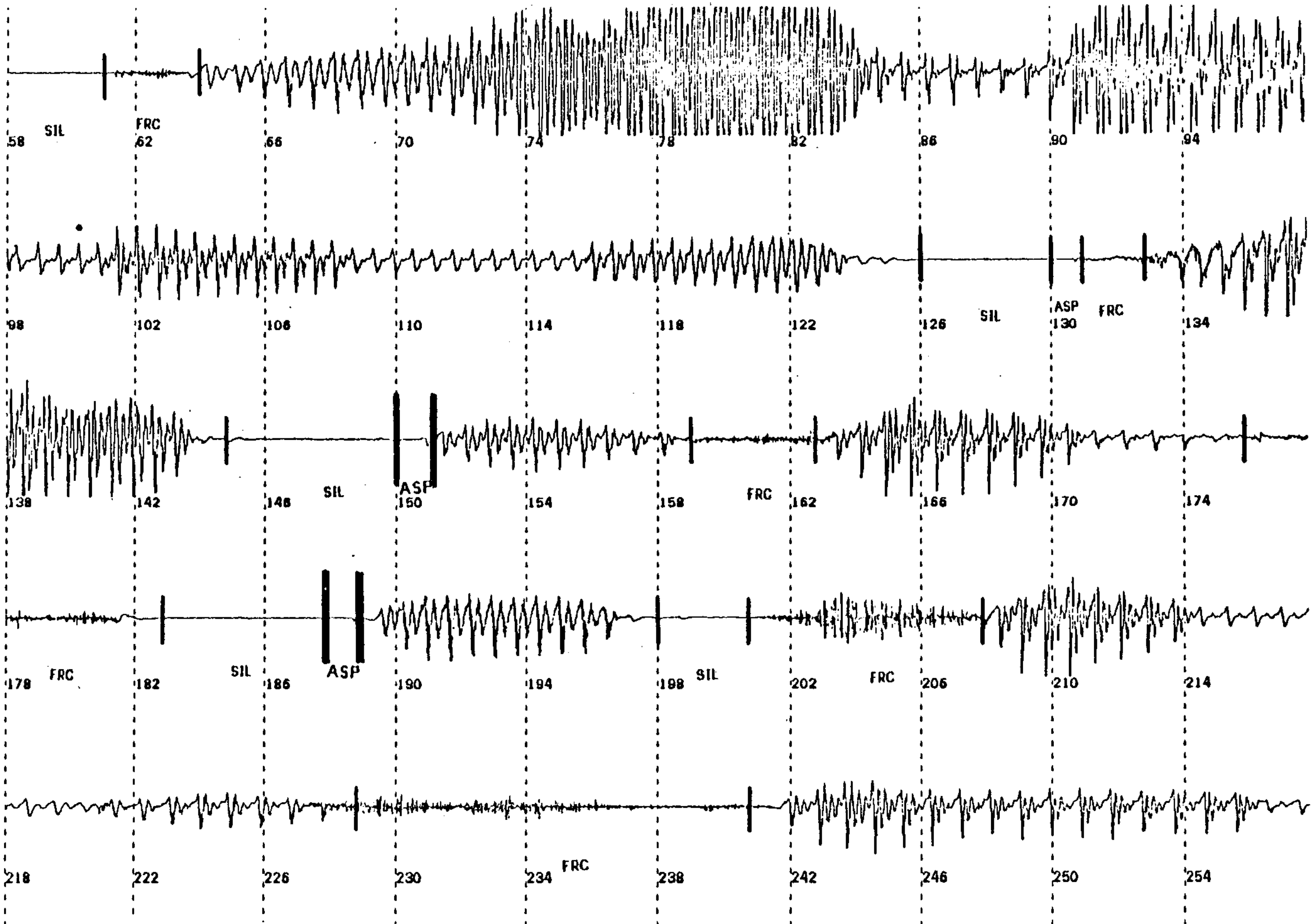

Figure 14: Result of application of decision rule D6 LAD.ADC LAD.UTT 30-May-78 12:49 UTT #: 3 PAGE 1 OF 2

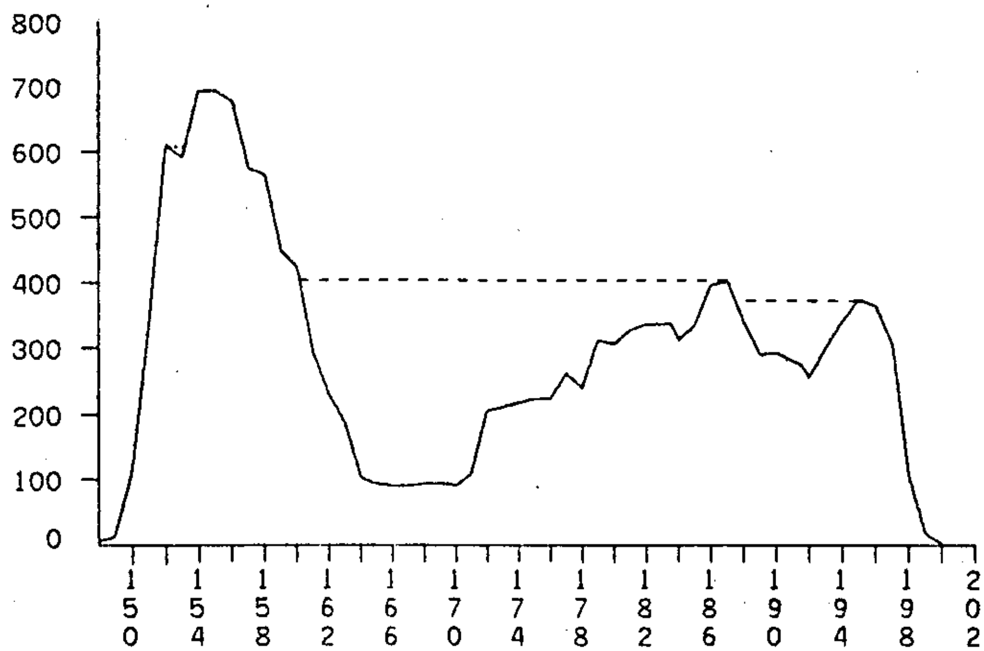

Figure 15a: Curve of SP levels with convex hull

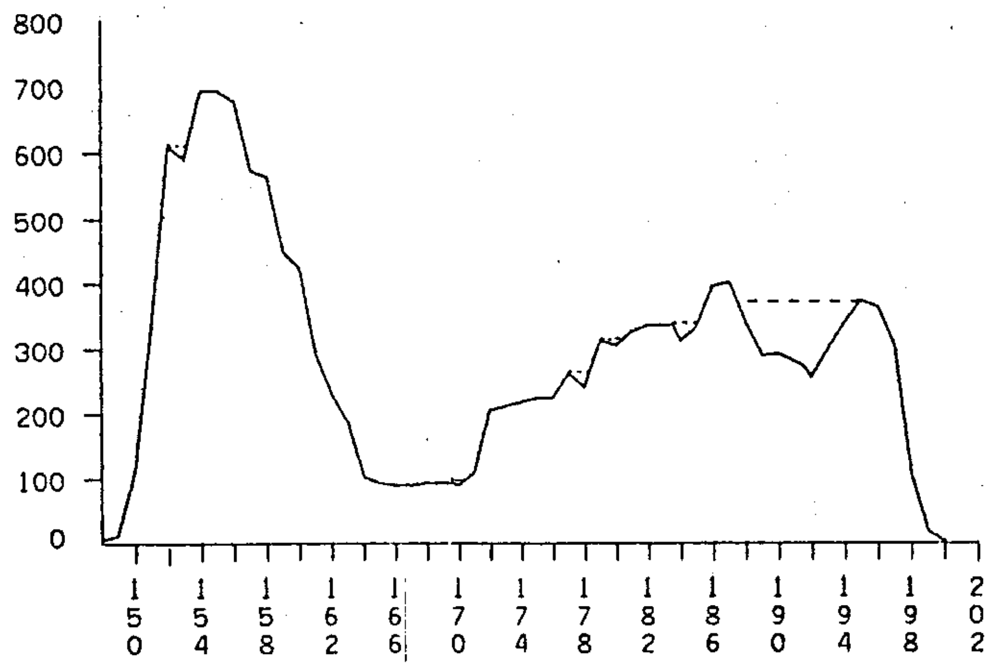

Figure 15b: Stage two of convex hull operation

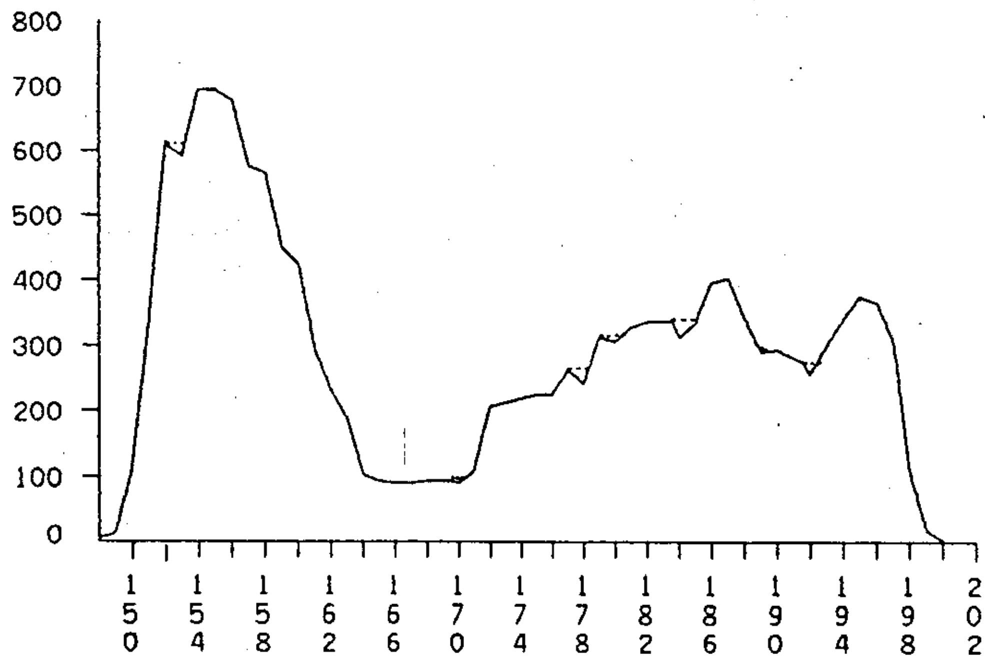

Figure 15c: Curve of SP levels with completed convex hull operation

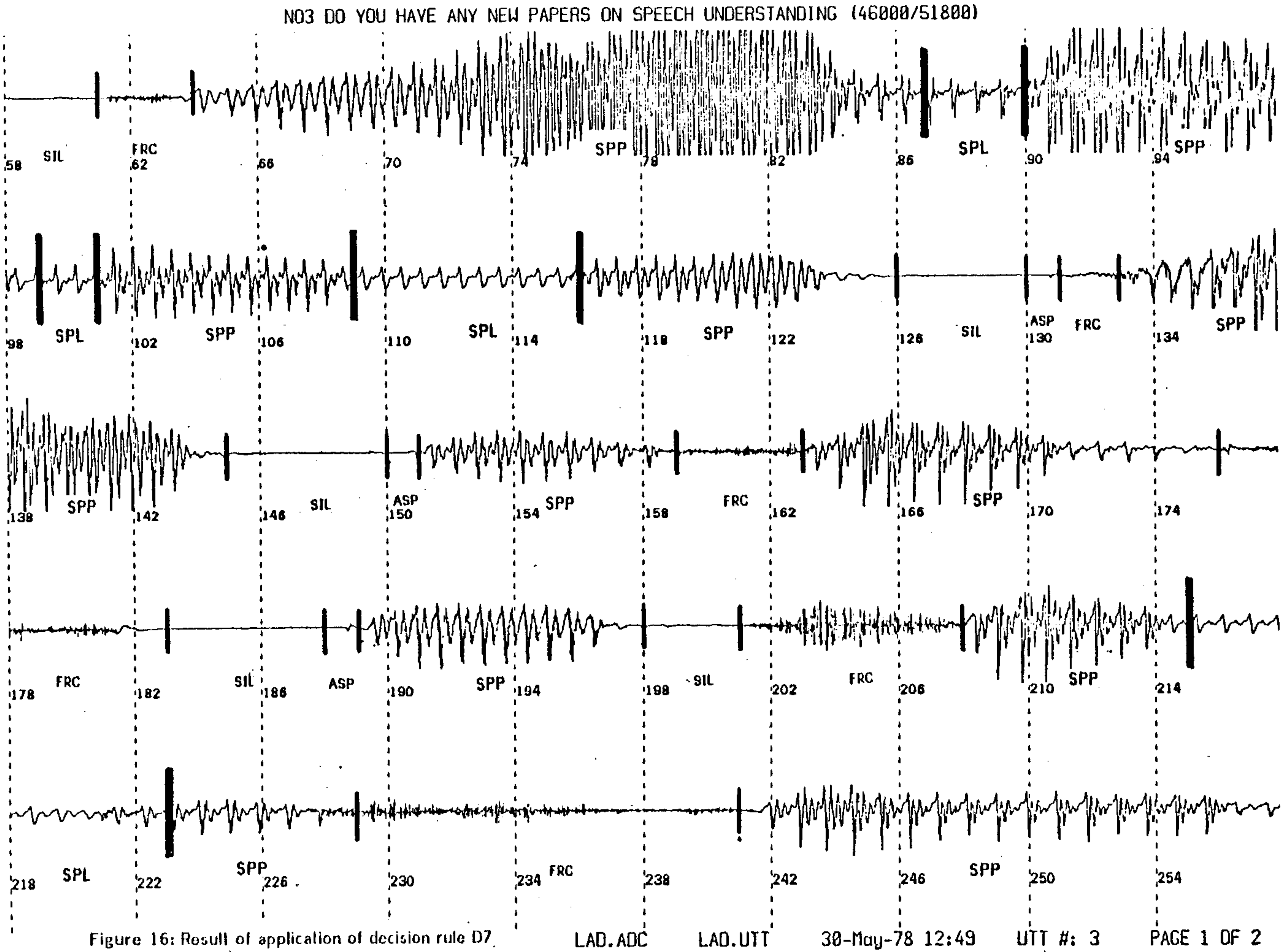

Figure 16: Result of application of decision rule D7

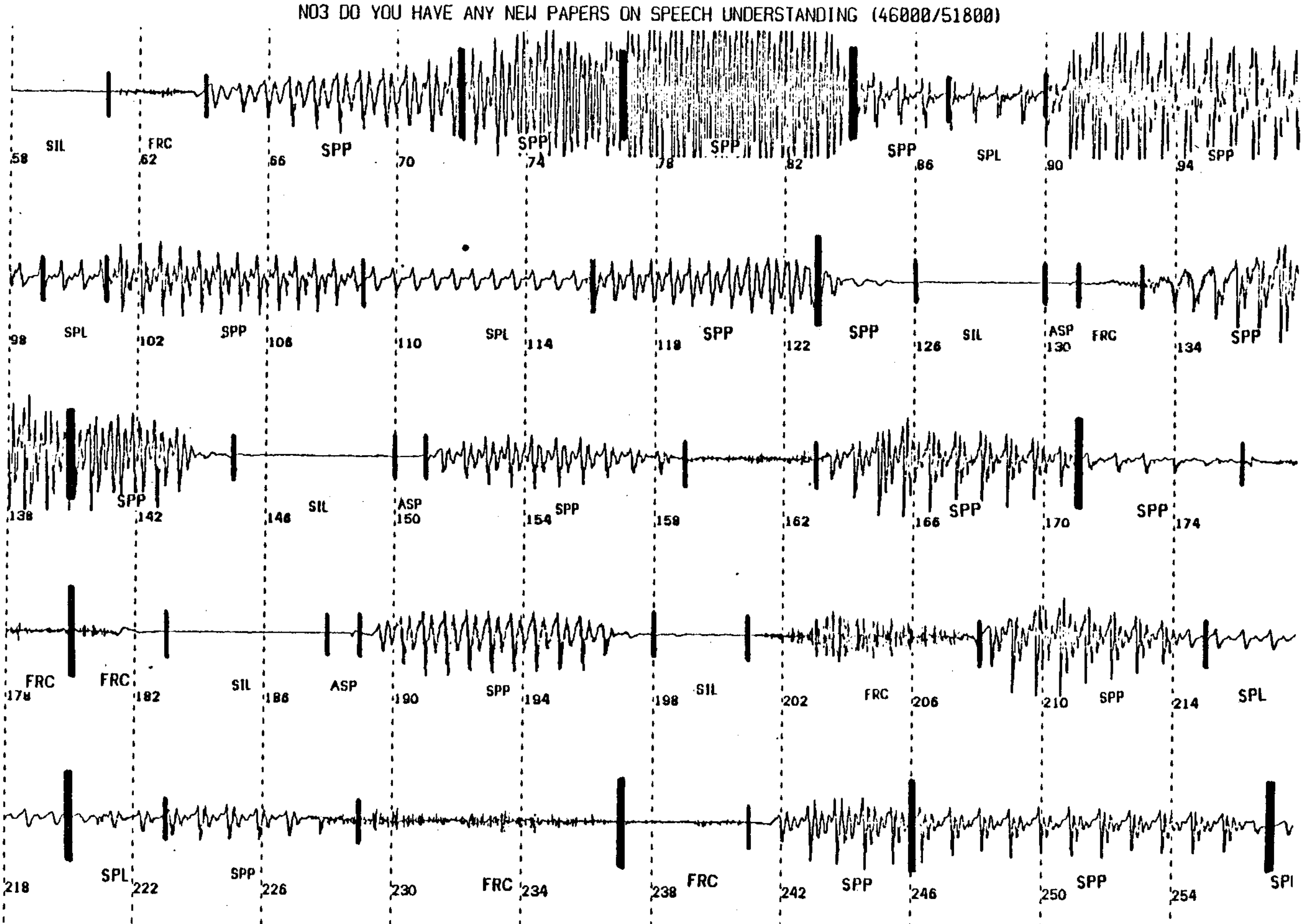

Figure 17: Result of application of decision rule D8

LAD.ADC LAD.UTT 30-May-78 12:49 UTT #: 3 PAGE 1 OF 2

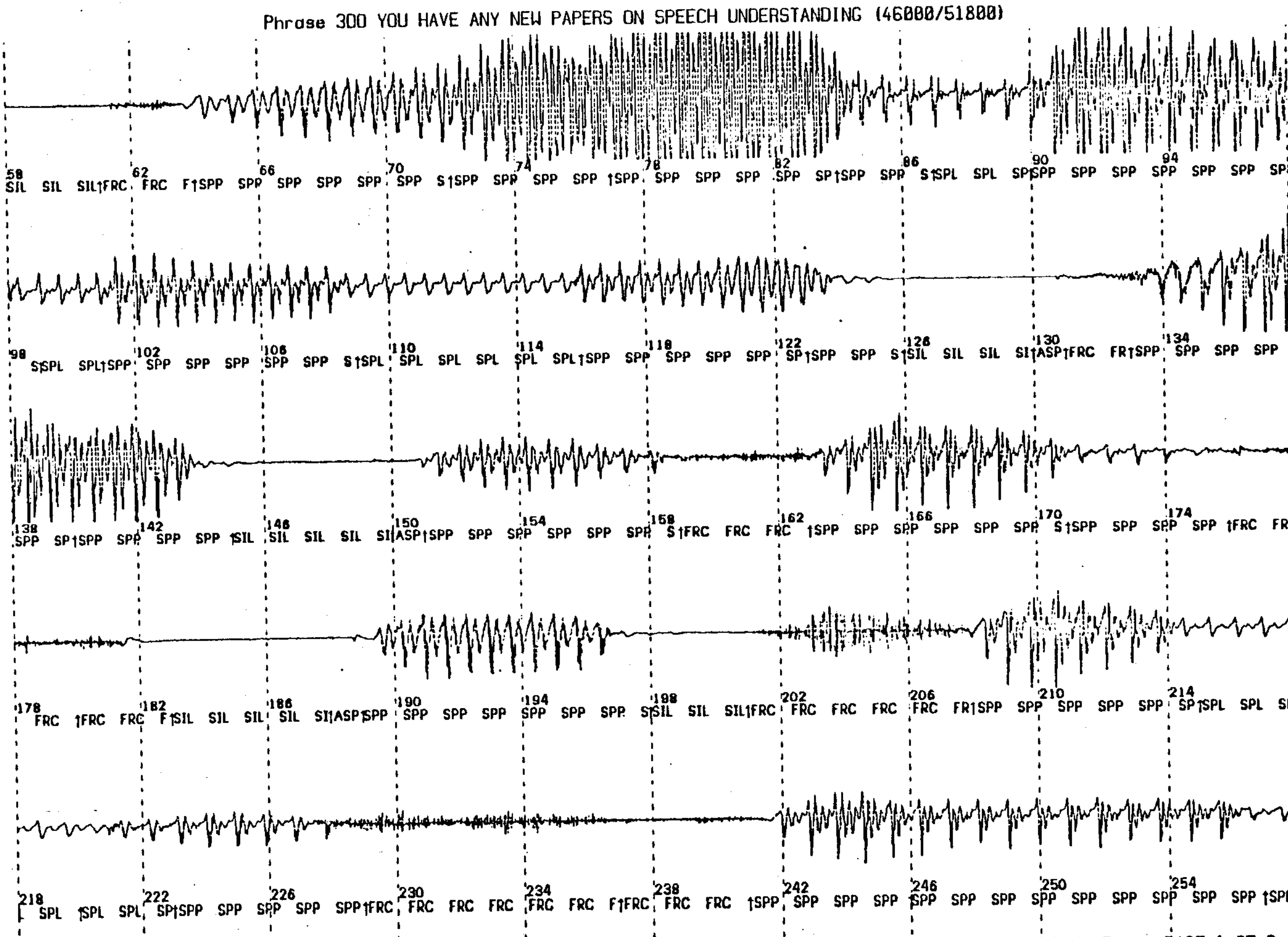

Figure 18: Final segmentation

Phrase 8WHAT IS THE TITLE OF THAT PAPER (176400/181800)

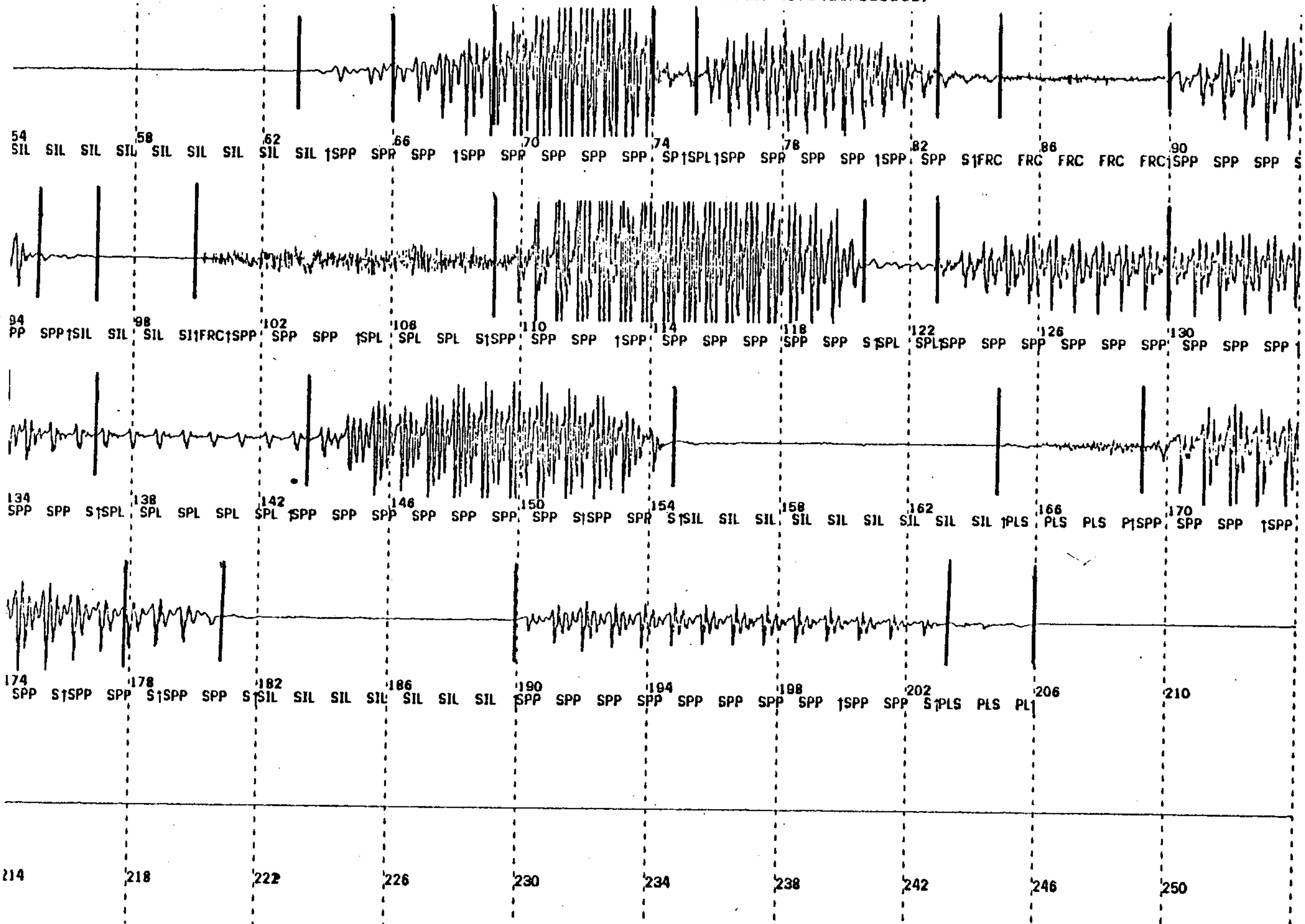

Figure 19: Example of non-critical segmentation errors LAD.ADC LAD450.SUT 18-Apr-78 13:17 UTT #: 8 PAGE 1 OF 1

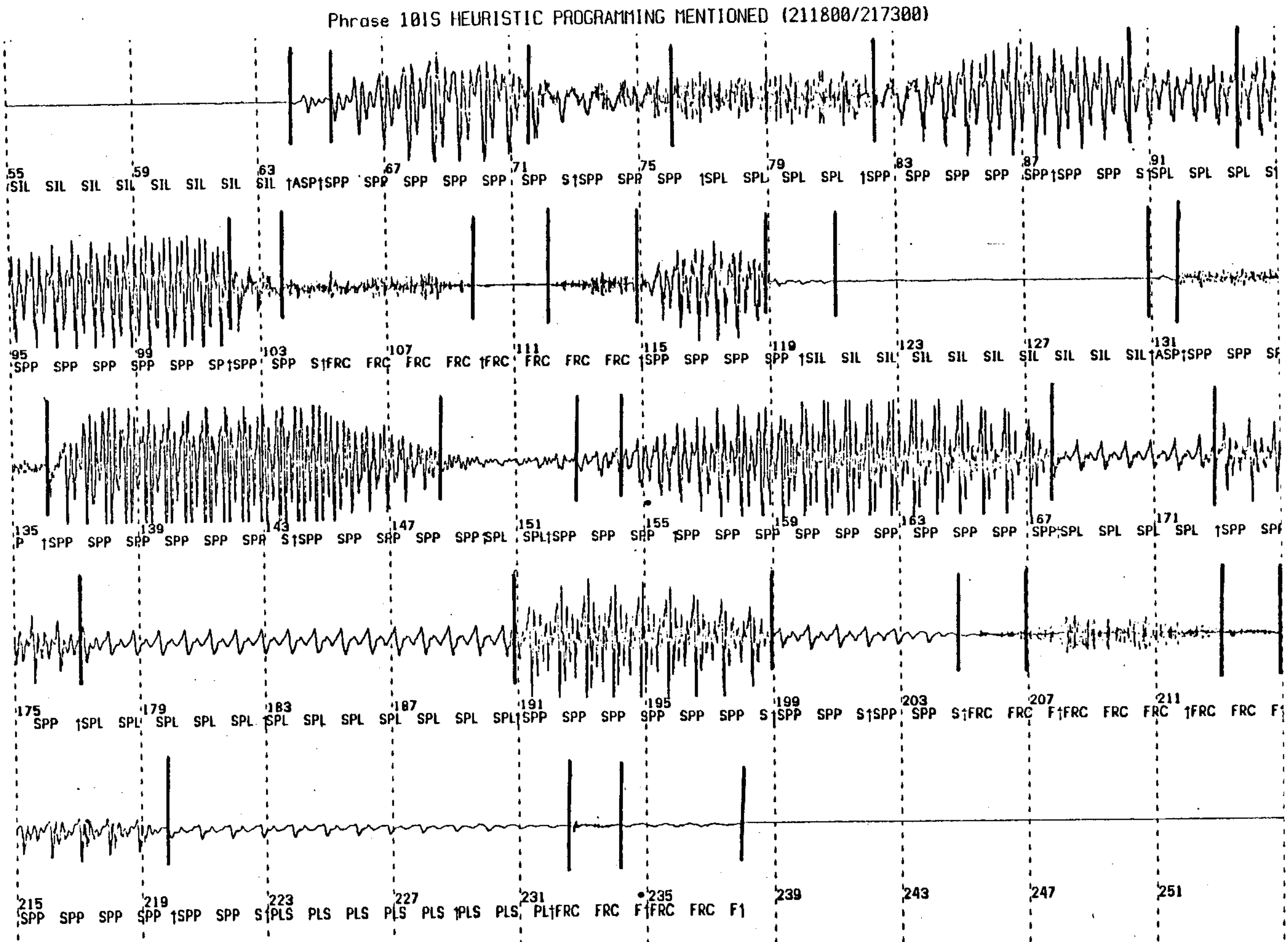

Figure 20: Example of critical segmentation errors

Appendix A: Program Listing

```
entry ;
begin "segmentor"
require "baysal.sai[a710sa80]" sourcefile;

! this is SEGMNT.sai, the segmentor for HEARSAY II and HARPY
! author G S GILL
! ***************
```

Directory of Segmentation procedures:

The first three procedures relate to the extraction of the ZAPDASH paramters, the remaining procedures are part of the segmentor.

```
SSIN(reference integer BUFPTR,DEPS!,SEPS!,BIAS! integer SETFLG)
          Semi-Static Init. Initialize dcbias & width of zero band from initial 'silence'.

GETSTATS(reference integer BUFPTR,DP,DZ,SP,SZ)
          returns the 4 ZAPDASH parameters for one frame of 100 samples starting at BUFPTR.

    STATS(reference integer BUFPTR,DMAX,DMIN,DZ,SMAX,SMIN,SZ)
          ZAPDASH parameter extraction procedure.  Takes 50 samples pointed to by BUFPTR and
          returns the maximum and minimum levels of differenced and smoothed data.  Also
          returns the number of zero crossings for each.  Written in assembler for speed.


SEGMENT(integer UTTBEG,UTTEND reference integer RNUM integer array RBEG,RTYP,PARS)
          This is the procedure which follows the flow chart in figure 5. The correction
          rules are implemented in-line, the other rules by procedure calls.

        COMSEG(integer STRT,FIN,DETCOD,REJCOD boolean procedure NECESS,SUFFIC)
          Called by SIL FRC and PLS detectors, COMSEG splits the segment starting at
          frame STRT through frame FIN into DETECTED(SIL,FRC,PLS) and REJECTED(NIL)
          types by testing the double thresholds represented by the procedures NECESSary
          and SUFFICIent.

    SILDET(integer STRT,FIN)
          Divide a segment (actually the whole utt) into silence and non-silence segments.
          A sub-segment is labelled silence iff each of its necessary and sufficient
          conditions are met.

    UFRDET(integer STRT,FIN)
          Divide an unlabeled segment into fricative and non-fricative segments.  A
          sub-segment is labelled FRC iff each of its necessary and sufficient conditions
          are met.

    PLSDET(integer STRT,FIN)
          Divide an unlabelled segment into low amplitude (PLS) and non-PLS segments.  A
          sub-segment is labelled PLS iff each of its necessary and sufficient conditions
          are met.

    ASPDET(integer STRT,FIN,LTYP)
          Called for each NIL seg before SPDIPS.  If the type of the preceeding segment
          (LTYP) is  SIL check for an isolated peak within 1 frame of STRT.  If found
          label it ASP.
```

SPDIPS(integer STRT,FIN,LTYP)
Split the unlabeled segment from frame STRT through frame FIN into peaks and dips in the SP ZAPDASH parameter. Split only if the segment is over 50msec long.

VEXHULL(integer ST,FI,DIPMIN,DIPPCNT integer array F)
Convex hull algorithm. It smooths the signal by selecting only dips which are significant snough to label.

DIPDET(integer STRT,FIN,DETCOD,REJCOD integer array FUNC integer TALDUR)
Detect dips in convex hull, and segment between each dip-peek and peek-dip pair.

FPEAK(integer S,F integer array V)
Finds the first peak in convex hull between frames S and F.

FDIP(integer S,F integer array V)
Finds the first dip in convex hull between frames S and F.

DIPSEG(integer P,D integer array F)
Segment between peek and dip at point of max slope or point where value goes over 150% of dip or over average of peek and dip values whichever is closest to dip.

REGSPLIT(integer STRT,FIN,TYPE reference integer SNUM)
If the segment from frame STRT through FIN is over 60 msec, check the spread of the ZAPDASH parameters from frame to frame. If the levels change significantly, split the segment.

GETCUR(integer FRAME)
If FRAME is 0 initialize min/max values to zero, otherwise get the ZAPDASH paramters for this frame and update the min/max values.

COMPARE(integer LOC,STRT)
Compare the maximum levels of ZAPDASH parameters with the minimum * PRCNT. If the levels are sufficiently spread to force a segment boundary, return an indicator of which parameter forced the boundary.

SEGAT(integer STRT,FIN,TYP,TYPE reference integer SNUM)
Levels of the 'TYP'th ZAPDASH parameter between STRT and FIN were found to be sufficiently different to force a boundary. SEGAT decides where to place that boundary by looking for the frame with the largest slope of the ZAPDASH parameter which caused the split and examing the slopes of the other 3 parameters for a small region around that frame.

```
require "!|()" delimiters;
ifcr not declaration(notlive!version) thenc define notlive!version=true; endc
!       save space if compiling only a live demo version ;

define  MAXSEG = 200,   MAXFRAMES = 1000;

!               thresholds ;

define  SILSPHI=15,       SILSPLOW=7,       SILDZHI=20,       SILDZLOW=10,
        UFRSPHI=40,       UFRSPLOW=20,      UFRDZHI=45,       UFRDZLOW=25,
        PLSSPHI=51,       PLSSPLOW=25,      PLSDPHI=51,       PLSDPLOW=25,
        SPDMIN=20,        SPDPCNT=70,       SPTALD=3,         SPMINL=5,
        ASPSPMAX=100,     ASPDSMIN=175,     ASPMINL=2;

! definitions of ZAPDASH indexes and segment types ;
define MAX!SEG!NAM=7;
define  DPDEX=1,          DZDEX=2,          SPDEX=3,          SZDEX=4;
define  NIL! = 1,         SIL! = 2,         FRC! = 3,         SPP! = 4,
        PLS! = 5,         ASP! = 6,         SPL! = 7;

safe internal integer array PARS[1:MAXFRAMES,1:4];     ! holds ZAPDASH parameters ;
safe internal integer array AVPARS[0:MAXSEG,1:4];      ! holds average ZAPDASH pars for each segment ;
safe internal integer array SPHULL[1:MAXFRAMES];       ! holds convex hull ;
internal integer SP90,DP90;                            ! 90 %ile values of SP and DP ;
safe integer array SBEG,STYP[1:MAXSEG];                ! begin time and type of each segment ;
internal integer UTTBEG,OFFSET;                        ! begin time of utterance ;
internal integer SNUM;                                 ! segment number ;
boolean SIL!FRST,UFR!FRST,PLS!FRST,SPD!FRST;           ! used in dump!mode ;

define dump!mode=true;
ifcr not declaration(dump!mode) thenc define dump!mode=false; endc
define dumpout(flag,x)=
   |ifcr dump!mode
    thenc if (dumpall and flag and chd>0) then out(chd,x);
    elsec
    endc
   |;

ifcr dump!mode
thenc internal boolean CHD,CHP,CHF,DUMPALL,DUMPARS;      ! channels and flags for dumps ;
      preload!with "out!of!range","NIL","SIL","FRC","SPP","PLS","ASP","SPL";
      string array SEGNAMES[0:MAX!SEG!NAM];
endc
```

```
!       ZAPDASH calculations
                                        ! save min/max levels of differenced and smoothed samples ;
;
integer DOMAX,DOMIN,SOMAX,SOMIN;
integer DSTATE,SSTATE;  ! save state for zero crossing counts ;
integer DEPS,SEPS;        ! differenced,smoothed epsilons ;
integer DZERO,DMZERO,SZERO,SMZERO;        ! zero band levels(M=minus ;
integer BIAS;    ! dcbias = average of several silence frames ;
integer SMA,SMB;          ! intermediate values for smooth calculation ;
integer LASTPT;           ! save last sample ;
safe internal integer array SPHISTO,DPHISTO[0:512];      ! for calculation of %iles ;

! ******************************************************************************* ;
internal simple procedure SSIN(reference integer BUFPTR,DEPS!,SEPS!,BIAS!; integer SETFLG);
! ******************************************************************************* ;
begin "ssin"
!       semi-static init. initialize dcbias & width of zero band from initial 'silence' ;
    integer BYTE,DMIN,SMIN,DMAX,SMAX,I;
    if SETFLG
    then begin
        DEPS←(DEPS!+1) ash -1; ! amount deleted from peek-peek calculation ;
        DZERO←(DEPS!+2) ash -2; ! +/- zero band epsilons ;
        DMZERO←-DZERO;
        SEPS←(SEPS! +1) ash -1;
        SZERO←(SEPS!+2) ash -2;
        SMZERO←-SZERO;
        BIAS←BIAS!;
        DSTATE←SSTATE←1;          ! initialize states for zero crossing counts ;
        arrclr(SPHISTO);          ! initialize histogrammes to zero ;
        arrclr(DPHISTO);
        DOMIN←DOMAX←SOMIN←SOMAX←0;     ! init old min/max to silence ;
        return;
        end;

    LASTPT←DMAX←SMAX←SMA←SMB←0;
    DMIN←SMIN←500;
    define NUMSAMPLES=100;            ! 100 samples per frame (10 m sec) ;
    start!code "sumloop"              ! calculate bias (av over all samples) ;
        define PTR=1, CNT=2, A=3, B=4;
                                   !    for I←1 thru NUMSAMPLES do BIAS!←BIAS!+ildb(PTR);
        LABEL BAK;                 !    BIAS!←(BIAS! + NUMSAMPLES%2) div NUMSAMPLES;
        MOVE    PTR,BUFPTR;
        MOVEI   CNT, NUMSAMPLES;
        MOVEI   B, NUMSAMPLES%2;           ! for rounding ;
    BAK: ILDB   A,PTR;
        ADD     B,A;
        SOJG    CNT, BAK;
        IDIVI   B,NUMSAMPLES;
        MOVEM   B,BIAS!;
    end "sumloop";
```

```
for i←1 thru NUMSAMPLES%5 do          ! calculate 1 smooth(-1 0 1 2 4 4 4 2 1 0 -1) ;
begin                                  ! sample for every 5th original sample ;
    define GETPOINT=
            [BYTE←ildb(BUFPTR)-BIAS];         ! get sample & subtract dcbias ;
            LASTPT←BYTE-LASTPT;              ! calculate difference sample ;
            DMIN←DMIN min LASTPT;             ! save min/max diff value over frame ;
            DMAX←DMAX max LASTPT;
            LASTPT←BYTE;
            ];
    GETPOINT;
    SMA←SMA+4*BYTE;
    GETPOINT;
    SMA←SMA+2*BYTE;
    SMB←SMB+BYTE;
    GETPOINT;
    SMA←SMA+BYTE;
    SMB←SMB+2*BYTE;
    GETPOINT;
    SMB←SMB+4*BYTE;
    GETPOINT;
    SMA←(SMA-BYTE);
    SMIN←SMIN min SMA;
    SMAX←SMAX max SMA;
    SMA←SMB+4*BYTE;
    SMB←-BYTE;
end;
DEPS!←DMAX-DMIN;     ! range of differenced samples ;
SEPS!←(SMAX-SMIN+8) ash -4; ! divide by sum of filter(16) ;
end"ssin";
```

```
! ************************************************************************** ;
internal simple procedure STATS(reference integer BUFPTR,DMAX,DMIN,DZ,SMAX,SMIN,SZ);
! ************************************************************************** ;
begin "stats"
!        ZAPDASH parameter extraction procedure.  Takes 50 samples pointed
!        to by bufptr and return max & min levels of differenced and
!        smoothed data.  Also returns the number of zero crossings
!        for each.  Written in assembler for speed.
;
startcode "squees"
label LOOP,L11,L21,L12,L22,L13,L23,L14,L24,L15,L25,L16,L26;
! assign accumulators ;
define  PTR=0, A=1, B=2, TMP=3, MAXD=4, MIND=5, MAXS=6, MINS=7,
        ACBIAS='10, I='11, SA='13, SB='14;
redefine NUMSAMPLES=50; ! half frame (5 m sec) ;
define LAB=0;
define  ZEROXINGS(AC,L)=
       { redefine LAB=LAB+1;     ! create unique labels ;
         redefine L1="L1"&cvms(LAB);
         redefine L2="L2"&cvms(LAB);
         redefine LZ="L"&"Z";    ! translates to DZ, SZ ;
         redefine LZERO="L"&"ZERO"; ! DZERO, SZERO ;
         redefine LMZERO="L"&"MZERO";    ! DMZERO, SMZERO ;
         redefine STATE="L"&"STATE";     ! DSTATE, SSTATE ;
        CAML    AC, LMZERO;
        JRST    L1;
        SKIPG   STATE;  ! > 0 if approaching 0 from above else < 0 ;
        JRST    L2;
        MOVNS   STATE;
        AOSA    LZ;
L1:     CAMG    AC, LZERO;
        JRST    L2;
        SKIPL   STATE;
        JRST    L2;
        MOVNS   STATE;
        AOS     LZ;
L2:
       };

define  MINMAX(AC,L)=
       { redefine MAXL="MAX"&"L";          ! MAXD, MAXS ;
         redefine  MINL="MIN"&"L";         ! MIND, MINS ;
         CAMLE   AC, MAXL;
         MOVEM   AC, MAXL;
         CAMGE   AC, MINL;
         MOVEM   AC, MINL;
       };

redefine GETPOINT=
       { ILDB    A, PTR;            ! get point ;
         SUB     A,ACBIAS;                  ! convert from excess '400 ;
         SUBM    A, B;              ! B=A-B, B is previous point ;
         MINMAX(B,D);               ! min,max differences ;
         ZEROXINGS(B,D);            ! # zero crossings for differences ;
         MOVE    B, A;              ! save this point for next difference ;
       };
```

```
        MOVE    PTR, BUFPTR;
        SETZB   MAXD, MAXS;
        SETZB   MIND, MINS;
        MOVE    SA, SMA;
        MOVE    SB, SMB;
        MOVE    B, LASTPT;
        MOVE    ACBIAS, BIAS;
        MOVEI   I, NUMSAMPLES%5;
LOOP:
        GETPOINT;               ! get first point ;
        ASH     A, 2;
        ADD     SA, A;          ! + 4 * A ;

        GETPOINT;               ! get second point ;
        add     SB, A;
        ASH     A, 1;
        ADD     SA, A;          ! + 2 * A ;
        GETPOINT;               ! get third point ;
        ADD     SA, A;          ! + 1 * A ;
        ASH     A, 1;
        ADD     SB, A;          ! + 2 * A ;

        GETPOINT;               ! fourth ;
        ASH      A, 2;
        ADD     SB, A;

        GETPOINT;               ! get fifth point ;
        SUB     SA, A;
        ASH     SA, -4;         ! divide by 16 ;
        MINMAX(SA,S);           ! min,max of smoothed points ;
        ZEROXINGS(SA,S);        ! # zero crossings for smoothed points ;
        MOVE    SA, SB;
        MOVN    SB, A;
        ASH      A, 2;
        ADD     SA, A;

        SOJG    I,LOOP;         ! go get another 4 points till you get a hundred ;

        MOVEM   SA, SMA;
        MOVEM   SB, SMB;
        MOVEM   B, LASTPT;
        MOVEM   PTR, BUFPTR;
        MOVEM   MIND, DMIN;
        MOVEM   MAXD, DMAX;
        MOVEM   MINS, SMIN;
        MOVEM   MAXS, SMAX;
end "squees";

end "stats";
```

```
! ************************************************************************ ;
internal simple procedure GETSTATS(reference integer BUFPTR,DP,DZ,SP,SZ);
! ************************************************************************ ;
begin "getstats"
!       return ZAPDASH parameters for 100 samples starting at bufptr.

;
    integer DRMAX,DLMAX,DRMIN,DLMIN,SRMAX,SLMAX,SRMIN,SLMIN;
    STATS(BUFPTR,DRMAX,DRMIN,DZ←0,SRMAX,SRMIN,SZ←0);    ! calc 1st half of frame ;
    STATS(BUFPTR,DLMAX,DLMIN,DZ,SLMAX,SLMIN,SZ);         ! calc 2nd half of frame ;
    DP←((DOMAX max DRMAX max DLMAX)
         -(DOMIN min DRMIN min DLMIN)-DEPS) max 0;    ! max of last 3 half frames ;
    addl(DPHISTO[DP min 512]);              ! incr histogramme ;
    SP←((SOMAX max SRMAX max SLMAX)
         -(SOMIN min SRMIN min SLMIN)-SEPS) max 0;    ! min of last 3 half frames ;
    addl(SPHISTO[SP min 512]);
    DOMAX←DLMAX;                ! remember last half frame for next frame calculation ;
    DOMIN←DLMIN;
    SOMAX←SLMAX;
    SOMIN←SLMIN;
 end "getstats";


 ! ****************************************************** ;
 internal integer simple procedure DP!SP!RATIO(integer I);
 ! ****************************************************** ;
 return(((PARS[I,DPDEX] ash 7) max 1) div (PARS[I,SPDEX] max 1));
```

```
! ************************************************************************************* ;
simple procedure COMSEG(integer STRT,FIN,DETCOD,REJCOD; boolean procedure NECESS,SUFFIC);
! ************************************************************************************* ;
begin "comseg"
!       called by SIL FRC and PLS detectors. COMSEG splits
!       the segment starting at frame STRT thru frame FIN
!       into DETECTED(SIL,FRC,PLS) and REJECTED(NIL) types
!       by testing the double thresholds represented by the
!       procedures NECESSary and SUFFICIent.
;
    integer SEGIS;      ! this frame meets NECESS conditions ;
    integer SEGWAS;     ! last frame met NECESS conditions ;
    integer DETECT;     ! SUFFIC conditions are met ;
    integer TBEG;       ! start of DETECT segment if SUFFIC conditions become met ;
    integer T;  ! frame counter ;
    ifcr dumplmode
    thenc string rejnam,detnam;
         if dumpall then begin rejnam←" "&segnames[rejcod]&" ";
                               detnam←" "&segnames[detcod]&" ";
                          end;
    endc

    ! Assume beginning REJECT segment;
    SBEG[SNUM←SNUM+1]←STRT;
    STYP[SNUM]←REJCOD;
    SEGWAS←DETECT←false;
    ! Initialize tests (in case they have OWNs);
    SUFFIC(0); NECESS(0);
    dumpout(true,cvs(strt+offset))

    for T←STRT thru FIN-1 do
    begin "tloop"
        SEGIS←NECESS(T);
        if SEGIS then
        begin "detected"
            if not SEGWAS then TBEG←T;  ! this is first frame to satisfy NEC condition ;
            if not(DETECT) and SUFFIC(T) then
            begin
                if TBEG>STRT then add1(SNUM);   ! if at strt label 1st seg as detected ;
                SBEG[SNUM]←TBEG;        ! else create new seg with DETECT label ;
                STYP[SNUM]←DETCOD;
                dumpout(true,rejnam&cvs(tbeg+offset))
                DETECT←true;
            end;
        end "detected"
        else
        begin "rejected"
            if not SEGWAS then continue "tloop";
            SUFFIC(0); NECESS(0);       ! reset NEC and SUF to false ;
            if DETECT then
            begin
                SBEG[add1(SNUM)]←T;     ! create new seg with REJECT label ;
                STYP[SNUM]←REJCOD;
                dumpout(true,detnam&cvs(t+offset))
                DETECT←false;
            end;
        end "rejected";
        SEGWAS←SEGIS;
    end "tloop";
    SBEG[SNUM+1]←FIN;
    dumpout(true,((if detect then detnam else rejnam)&cvs(fin+offset)&crlf))
end "comseg";
```

```
    ! Specific Detection Procedures;

! ******************************** ;
procedure SILDET(integer STRT,FIN);
! ******************************** ;
begin "sildet"
!       divide a segment (actually the whole utt) into silence &
!       non-silence segments.  A sub-segment is labelled silence
!       iff each of these conditions are met:
!           a) at least one frame has SP ≤ SILSPLOW(7)
!           b) at least one frame has DZ ≤ SILDZLOW(10)
!           c) each frame has SP ≤ SILSPHI(15)
!           d) each frame has DZ ≤ SILDZHI(20)
;
        boolean procedure NECESS(reference integer T);
        if T>0 then return(
        (PARS[T,SPDEX] leq SILSPHI) and
        (PARS[T,DZDEX] leq SILDZHI))
        else return(false);

        boolean procedure SUFFIC(reference integer T);
        begin "suffic"
            own boolean SPGOOD,DZGOOD;
            if T leq 0 then return(SPGOOD←DZGOOD←false);
            SPGOOD←SPGOOD or (PARS[T,SPDEX] leq SILSPLOW);
            DZGOOD←DZGOOD or (PARS[T,DZDEX] leq SILDZLOW);
            return(SPGOOD and DZGOOD);
        end "suffic";

    dumpout(sil!frst,("detecting SIL SP-("&cvs(silsphi)&","
            &cvs(silsplow)&") & DZ-("&cvs(sildzhi)&","&cvs(sildzlow)&")"&crlf))
    COMSEG(STRT,FIN,SIL!,nil!,NECESS,SUFFIC);
    SIL!FRST←false;
 end "sildet";
```

```
! ******************************** ;
procedure UFRDET(integer STRT,FIN);
! ******************************** ;
begin "ufrdet"
!       divide an unlabeled segment into fricative and
!       non-fricative segments.  A sub-segment is labelled FRC
!       iff each of these conditions are met:
!           a) at least one frame has SP ≤ UFRSPLOW(20)
!           b) at least one frame has DZ ≥ UFRDZHI(45)
!           c) each frame has SP ≤ UFRSPHI(40)
!           d) each frame has DZ ≥ UFRDZLOW(25)
;
        boolean procedure NECESS(reference integer T);
        if T>0 then return(
        (PARS[T,SPDEX] leq UFRSPHI) and
        (PARS[T,DZDEX] geq UFRDZLOW))
        else return(false);

        boolean procedure SUFFIC(reference integer T);
        begin "suffic"
            own boolean SPGOOD,DZGOOD;
            if T leq 0 then return(SPGOOD←DZGOOD←false);
            SPGOOD←SPGOOD or (PARS[T,SPDEX] leq UFRSPLOW);
            DZGOOD←DZGOOD or (PARS[T,DZDEX] geq UFRDZHI);
            return(SPGOOD and DZGOOD);
        end "suffic";

    dumpout(ufr!frst,("detecting UFR SP-("&cvs(ufrsphi)&","
            &cvs(ufrsplow)&") & DZ-("&cvs(ufrdzhi)&","&cvs(ufrdzlow)&")"&crlf))
    COMSEG(STRT,FIN,FRC!,nil!,NECESS,SUFFIC);
    UFR!FRST←false;
end "ufrdet";
```

```
! ****************************************** ;
simple procedure ASPDET(integer STRT,FIN,LTYP);
! ****************************************** ;
begin "aspdet"
!       called for each NIL seg before SPDIPS.  If type of preceding
!       segment (LTYP) is SIL check for isolated peak within 1 frame
!       of STRT.  If found label it ASP.
;
    if STRT>2   ! not first segment ;
    and LTYP=SIL!
    and ((FIN-STRT) > ASPMINL)
    and (PARS[STRT,SPDEX] leq ASPSPMAX)
    then begin "detasp"
        integer D0,D1,D2,D3,D4,ASPT;
        D0←DP!SP!RATIO(STRT-2); ! calculate DP/SP ;
        D1←DP!SP!RATIO(STRT-1);
        D2←DP!SP!RATIO(STRT);
        D3←DP!SP!RATIO(STRT+1);
        D4←DP!SP!RATIO(STRT+2);
        if (D1 max D2 max D3) geq ASPDSMIN then
        begin
        ASPT←if D0<D1>D2 then strt+1 ! should really steal a cs from the previous seg;
        else if D1<D2>D3 then strt+1
        else if D2<D3>D4 then strt+2
        else 0;
        if ASPT > 0 then begin SBEG[add1(SNUM)]←STRT; SBEG[SNUM+1]←ASPT;
                STYP[SNUM]←ASP!;
                dumpout(true,(cvs(strt+offset)&" ASP "&cvs(aspt+offset)&tab&tab
                        &cvs(d0)&" "&cvs(d1)&" "&cvs(d2)&" "&cvs(d3)&" "&cvs(d4)&crlf))
                STRT←ASPT; end;
        end;
        end "detasp";
        SBEG[add1(SNUM)]←STRT;
        STYP[SNUM]←NIL!;
        SBEG[SNUM+1]←FIN;
  end "aspdet";
```

```
! ****************************** ;
procedure PLSDET(integer STRT,FIN);
! ****************************** ;
begin "plsdet"
!        divide an unlabelled segment into PLS and non-PLS segments.  A
!        sub-segment is labelled PLS iff each of these conditions are met:
!            a) at least one frame has SP ≤ PLSSPLOW(25)
!            b) at least one frame has DP ≤ PLSDPLOW(25)
!            c) each frame has SP ≤ PLSSPHI(51)
!            d) each frame has DP ≤ PLSDPHI(51)
;
        boolean procedure NECESS(reference integer T);
        if T>0 then return(
        (PARS[T,DPDEX] leq PLSDPHI) and
        (PARS[T,SPDEX] leq PLSSPHI))
        else return(false);

        boolean procedure SUFFIC(reference integer T);
        begin "suffic"
            own boolean SPGOOD,DPGOOD;
            if T leq 0 then return(SPGOOD←DPGOOD←false);
            SPGOOD←SPGOOD or (PARS[T,SPDEX] leq PLSSPLOW);
            DPGOOD←DPGOOD or (PARS[T,DPDEX] leq PLSDPLOW);
            return(SPGOOD and DPGOOD);
        end "suffic";

    dumpout(pls!frst,("detecting PLS SP-("&cvs(plssphi)&","
            &cvs(plssplow)&") & DP-("&cvs(plsdphi)&","&cvs(plsdplow)&")"&crlf))
    COMSEG(STRT,FIN,PLS!,NIL!,NECESS,SUFFIC);
    PLS!FRST←false;
end "plsdet";
```

```
! ********************************************************************* ;
procedure VEXHULL(integer ST,FI,DIPMIN,DIPPCNT; safe integer array F);
! ********************************************************************* ;
begin "vexhull"

!       Convex hull algorithm.  It smooths the signal by selecting only
!       dips which are significant snough to label ;

    integer I,V,L,DUM;
    safe integer array H[ST:(FI max ST)];

! find max value;
    L←ST; V←F[L];
    for I←ST+1 thru FI do if F[I]>V then V←F[L←I];

!   make hull;
    H[ST]←F[ST];
    for I←ST+1 thru L do H[I]←F[I] max H[I-1];
    H[FI]←F[FI];
    for I←FI-1 downto L do H[I]←F[I] max H[I+1];

    while true do
    begin "dips"
    ! find biggest dip;
    V←0; L←ST;
    for I←ST thru FI do if (DUM←H[I]-F[I])>V then begin V←DUM; L←I; end;
    if V=0 then done "dips";        ! all dips considered :: H=F;
    ! if significant, fixup hull, else bring up function to hull;
    if (V geq DIPMIN) and (F[L] leq (H[L]*DIPPCNT)/100)
    then begin "hulldown"
            H[L]←F[L];
            for I←L+1 thru FI do
            begin
                DUM←F[I] max H[I-1];
                if DUM geq H[I] then done else H[I]←DUM;
            end;
            for I←L-1 downto ST do
            begin
                DUM←F[I] max H[I+1];
                if DUM geq H[I] then done else H[I]←DUM;
            end;
         end "hulldown"
     else begin "funcup"
            dum←F[L]←H[L];
            for I←L+1 thru FI do if F[I]<dum then F[I]←dum else done;
            for I←L-1 downto ST do if F[I]<dum then F[I]←dum else done;
         end "funcup";
     end "dips";
     arrblt(F[ST],H[ST],FI-ST+1);
  end "vexhull";
```

```
! ************************************************************* ;
simple integer procedure DIPSEG(integer P,D; safe integer array F);
! ************************************************************* ;
    begin "dipseg"

!       segment between peek and dip at point of max slope
!       or point where value goes over 150% of dip
!       or over average of peek and dip values
!       whichever is closest to dip.
;
    integer FP,FD,T,TH,S,I,V,J,SL;
    ! Segment at threshold (2*dip min (dip+peak)/2) or greatest slope, whichever is closest to dip;
    FP←F[P]; FD←F[D];
    TH←(FD*1.5) min ((FD+FP)/2);
    if D>P then S←-1 else S←1;
    for T←D step S until P do if F[T]>TH then done;
    V←0;
    J←D;
    for I←D+S step S until P do
        begin "thrandmax"
        if (SL←F[I]-F[I-S]) > V then begin V←SL; J←I; end;
        end "thrandmax";
    ! T is thr loc J is slope loc, pick closest to D;
    if (s<0) then begin j←j+1; t←t+1; end;
    dumpout(true,("max slope@"&cvs(J+OFFSET)&";thre"&cvs(T+OFFSET)&"; 150%="&cvs(FD*1.5)
                &", av="&cvs((FD+FP)/2)&"]"&crlf))
    return(if S > 0 then (J min T) else (J max T));
    end "dipseg";

! ************************************************************* ;
simple integer procedure FPEAK(integer S,F; safe integer array V);
! ************************************************************* ;
    begin "fpeak"
!       finds first peak in convex hull between S & F ;

    integer I,J,K;
    if V[S+1]<V[S] then return(S);      ! peak at begining ;
    ! find rise;
    for I←S thru F-2 do if V[I+1] > V[I] then done;
    if I ≥ F-1 then return(F);  ! failed to find any peak ;
    ! find fall;
    for J←I thru F-1 do if V[J+1] < V[J] then done;
    ! find start of peak;
    for K←J downto I do if V[K-1] < V[K] then done;
    return((K+J)%2);
    end "fpeak";

! ************************************************************* ;
simple integer procedure FDIP(integer S,F; safe integer array V);
! ************************************************************* ;
    begin "fdip"
!       find first dip in convex hull between S & F ;

    integer I,J,K;
    ! find fall;
    for I←S thru F-1 do if V[I+1] < V[I] then done;
    ! find rise;
    for J←I thru F-1 do if V[J+1] > V[J] then done;
    ! find start of dip;
    for K←J downto I do if V[K-1] > V[K] then done;
    if J geq F then return(F);
    return((K+J)%2);
    end "fdip";
```

```
! ************************************************************************************ ;
procedure DIPDET(integer STRT,FIN,DETCOD,REJCOD; safe integer array FUNC; integer TALDUR);
! ************************************************************************************ ;
    begin "dipdet"
!       detect dips and segment between each dip-peak
!       and peak-dip pair
;
    integer I,T,P,D,U,V,X,LP,NS,NE;
    ifcr dump!mode
    thenc string detnam,rejnam;
         if dumpall then begin
                          rejnam←" "&segnames[rejcod]&" ";
                          detnam←" "&segnames[detcod]&" ";
                        end;
    endc

    SBEG[SNUM←SNUM+1]←STRT;
    STYP[SNUM]←REJCOD;

    NS←STRT+TALDUR; NE←FIN-TALDUR;
    ! Find first peak-dip or dip-peak pair;
    P←FPEAK(STRT,FIN,FUNC);
    D←FDIP(P,FIN,FUNC);
    ! loop alternates peaks and dips - segment between each;
    while true do begin "pdloop"
        if (P geq FIN) or (D geq FIN) then done "pdloop";
        if P<D then
            begin "peak-dip"
            T←DIPSEG(P,D,FUNC);
            if T geq (NS)
            then begin
            STYP[SNUM]←REJCOD;
            dumpout(true,cvs(sbeg[snum]+offset)&rejnam)
            SBEG[SNUM←SNUM+1]←T;
            STYP[SNUM]←DETCOD;
                end;
            P←FPEAK(D,FIN,FUNC);
            end "peak-dip"
        else begin "dip-peak"
            T←DIPSEG(P,D,FUNC);
            if (T leq (NE))
            then begin
                  STYP[SNUM]←DETCOD;
                  dumpout(true,cvs(sbeg[snum]+offset)&detnam)
                  SBEG[SNUM←SNUM+1]←T;
                  STYP[SNUM]←REJCOD;
                 end
            else SNUM←SNUM-1;
            D←FDIP(P,FIN,FUNC);
            end "dip-peak";
        end "pdloop";
    ! finish up;
    if STYP[SNUM]=DETCOD then SNUM←SNUM-1;   ! i.e. we ran out of room to find a final peak;
    SBEG[SNUM+1]←FIN;
    dumpout(true,cvs(sbeg[snum]+offset)
             &(if styp[snum]=rejcod then rejnam else detnam)
             &cvs(fin+offset)&crlf)
    end "dipdet";
```

```
! ****************************************** ;
simple procedure SPDIPS(integer STRT,FIN,LTYP);
! ****************************************** ;
    begin "spdips"
!       split segment from STRT thru FIN into peeks and dips.
!       If segment < SPMINL(5) dont split it.
;
    integer I;

dumpout(spd!frst,("son split: spdips: minlen="&cvs(spminl)
                &" dipmin="&cvs(spdmin)&" dippcnt="&cvs(spdpcnt)&crlf))
    SPD!FRST←false;
    If FIN-STRT < SPMINL or STRT ≤ 1 then begin
        SBEG[add1(SNUM)]←STRT; STYP[SNUM]←SPP!; SBEG[SNUM+1]←FIN; return; end;

    for I←STRT-1 thru FIN do SPHULL[I]←PARS[I,SPDEX];   ! create convex hull ;
    VEXHULL(STRT,FIN-1,SPDMIN,SPDPCNT,SPHULL);

    DIPDET(STRT,FIN,SPL!,SPP!,SPHULL,SPTALD);
    end "spdips";
```

```
!       REGION SPLITTING PROCEDURES
;
define  DPMIN!=10, DZMIN!=05, SPMIN!=10, SZMIN!=5, RGNSIZE=2;
ifcr dumpimode
thenc preload!with "","dp","dz","sp","sz";
      string array zapdashtyp[0:4];
endc
define MAXPRCNT=15;
preload!with 2.4, 2.3, 2.2, 2.1, 2.0, 1.9, 1.8, 1.7,1.6, 1.5, 1.4, 1.3, 1.2;
safe real array PRCNTABLE[1:MAXPRCNT];
safe integer array RGNHISTO[-RGNSIZE:RGNSIZE];
integer DPMAX,DZMAX,SPMAX,SZMAX,DPMIN,DZMIN,SPMIN,SZMIN; ! accumulate max/min ZAPDASH parameters ;
internal integer sgmdeb;                  ! debug flag ;

! ********************************** ;
simple procedure GETCUR(integer FRAME);
! ********************************** ;
begin "getcur"
!       if FRAME is 0 initialize min/max values, else recalculate min/max ZAPDASH levels
;
 integer DPCUR,DZCUR,SPCUR,SZCUR;       ! tmps to hold ZAPDASH pars ;
 if FRAME=0
 then begin
        DPMAX←DZMAX←SPMAX←SZMAX←0;
        DPMIN←DZMIN←SPMIN←SZMIN←400;
        return;
      end;
 DPMAX←DPMAX max (DPCUR←PARS[FRAME,DPDEX]);
 DPMIN←DPMIN min DPCUR;
 DZMAX←DZMAX max (DZCUR←PARS[FRAME,DZDEX]);
 DZMIN←DZMIN min DZCUR;
 SPMAX←SPMAX max (SPCUR←PARS[FRAME,SPDEX]);
 SPMIN←SPMIN min SPCUR;
 SZMAX←SZMAX max (SZCUR←PARS[FRAME,SZDEX]);
 SZMIN←SZMIN min SZCUR;
end "getcur";

! ********************************************** ;
integer simple procedure COMPARE(integer LOC,STRT);                ! COMPARE ;
! ********************************************** ;
begin "compare"
!       compare max levels with min*PRCNT to see if levels have sufficiently changed to force
!       segment boundary. return 0 if max < min*PRCNT else which ZAPDASH forced boundary.
;
real PRCNT;     ! hold calculated % to be used ;
integer typ;            ! hold the # of the ZAPDASH forcing boundary ;
integer t1,t2,t3,t4;    ! tmps for dump mode ;
PRCNT←PRCNTABLE[(LOC-STRT) min MAXPRCNT];
typ←0;
      if DPMAX > (t1←(DPMIN max DPMIN!)*PRCNT) then typ←1
 else if DZMAX > (t2←(DZMIN max DZMIN!)*PRCNT) then typ←2
 else if SPMAX > (t3←(SPMIN max SPMIN!)*PRCNT) then typ←3
 else if SZMAX > (t4←(SZMIN max SZMIN!)*PRCNT) then typ←4;
if (sgmdeb or typ)
then begin
        dumpout(true,(cvs(LOC+OFFSET)&"[%="&cvs(PRCNT*100)
                    &"] DP("&cvs(DPMIN)&","&cvs(DPMAX)&":"&cvs(t1)
                    &") DZ("&cvs(DZMIN)&","&cvs(DZMAX)&":"&cvs(t2)
                    &") SP("&cvs(SPMIN)&","&cvs(SPMAX)&":"&cvs(t3)
                    &") SZ("&cvs(SZMIN)&","&cvs(SZMAX)&":"&cvs(t4)
                    &(if typ then ")    break caused by "
                        &zapdashtyp[typ] else "")&crlf));

      end;
 return(typ);
end "compare";
```

```
! ******************************************************************** ;
simple procedure SEGAT(integer STRT,FIN,TYP,TYPE; reference integer SNUM);
! ******************************************************************** ;
begin "segat"
!       levels of the 'TYP'th ZAPDASH parameter between STRT and FIN
!       were found to be sufficiently different to force a boundary.
!       SEGAT decides where to place that boundary by looking for the
!       frame with the largest slope of the ZAPDASH parameter which
!       caused the split and examing the slopes of the other 3
!       parameters for a small region around that frame.
;
 integer A,B,SLOPE,DIFF,PT,PNT,I,J;

 dumpout(true,(zapdashtyp[typ]&" causing seg within ("&cvs(strt+offset)
        &" "&cvs(fin+offset)&") snum="&cvs(snum)))
 PT←((STRT+FIN) div 2) min (FIN-3);
 A←case TYP-1 of (PARS[PT,DPDEX],PARS[PT,DZDEX],PARS[PT,SPDEX],PARS[PT,SZDEX]);
 SLOPE←0;
 for I←PT+1 thru FIN do
 begin "maxslope"
  B←(case TYP-1 of (PARS[I,DPDEX],PARS[I,DZDEX],PARS[I,SPDEX],PARS[I,SZDEX]));
  DIFF←abs(B-A);
  if DIFF>SLOPE then begin SLOPE←DIFF; PT←I; end;
  A←B;
 end "maxslope";

dumpout(true,"slope["&cvs(slope)&"]@"&cvs(pt+offset)&crlf)
arrclr(RGNHISTO);
 RGNHISTO[0]←1;

 for j←1 thru 4
 do begin "otherfuncs"
    if j=TYP then continue;
    dum←(PT-RGNSIZE) max STRT;
  A←(case j-1 of (PARS[DUM,DPDEX],PARS[DUM,DZDEX],PARS[DUM,SPDEX],PARS[DUM,SZDEX]));
    SLOPE←0;
    PNT←0;
    for I←(PT-RGNSIZE+1) max (STRT+1) thru (PT+RGNSIZE)
    do begin
        B←(case j-1 of (PARS[I,DPDEX],PARS[I,DZDEX],PARS[I,SPDEX],PARS[I,SZDEX]));
        DIFF←abs(B-A);
        if DIFF>SLOPE then begin SLOPE←DIFF; PNT←I; end;
        A←B;
       end;
    if PNT>0 then add1(RGNHISTO[PNT-PT]);
 end "otherfuncs";
if dumpall and chd>0
then for i←-rgnsize thru rgnsize do out(chd," "&cvs(rgnhisto[i]));

 if RGNHISTO[0]≤1
 then for i← -RGNSIZE thru RGNSIZE
      do if RGNHISTO[i] ≥ 2 then begin PT←PT+I; done end;

 SBEG[add1(SNUM)]←PT;
 STYP[SNUM]←TYPE;
 dumpout(true," seg at "&cvs(pt+offset)&crlf)
end "segat";
```

```
! ******************************************************************** ;
simple procedure REGSPLIT(integer STRT,FIN,TYPE; reference integer SNUM);
! ******************************************************************** ;
begin "regsplit"
!       if segment > 60 m sec try spliting by checking spread
!       of ZAPDASH parameters over segment length
;
integer I,LOC,TYP;

 if sgmdeb then dumpout(true,("entered regsplit with("&cvs(strt+offset)
                        &" "&segnames[type]&" "&cvs(fin+offset)
                        &") snum="&cvs(snum)&crlf))
 SBEG[add1(SNUM)]←STRT;
 STYP[SNUM]←TYPE;
 if (FIN-STRT)<6 then begin SBEG[SNUM+1]←FIN; return; end;
STRT←STRT+2;
GETCUR(0);
GETCUR(STRT);
 for LOC←STRT+1 thru FIN-2
 do begin
        GETCUR(LOC);
        if TYP←COMPARE(LOC,STRT)
        then begin
                SEGAT(STRT,LOC,TYP,TYPE,SNUM);
                if (FIN-SBEG[SNUM]) < 3
                then begin
                        SBEG[SNUM]←FIN-3;
                        dumpout(true,"moving seg back to "&cvs(fin-3+offset)&crlf)
                     end;
                STRT←LOC←SBEG[SNUM]+2;
                GETCUR(0);
                GETCUR(STRT);
             end;
 end;
SBEG[SNUM+1]←FIN;
end "regsplit";
```

```
! ****************************************************************** ;
internal procedure SEGMENT(integer UTTBEG,UTTEND; reference integer RNUM;
! ****************************************************************** ;
                                safe integer array RBEG,RTYP;
                                safe integer array PARS);
    begin "segment"
      integer I;

      procedure copy(integer I);                           ! COPY ;
        begin "copy"
          RBEG[add1(RNUM)]←SBEG[I];
          RBEG[RNUM+1]←SBEG[I+1];
          RTYP[RNUM]←STYP[I];
        end;

      ! initial, null segmentation;
      RNUM←1; SNUM←0;
      RBEG[1]←UTTBEG;
      RBEG[2]←UTTEND;
      RTYP[1]←1;
        dumpout(true,crlf)


!       D1: SIL detection ;

      for I←1 thru RNUM do SILDET(RBEG[I],RBEG[I+1]);
      arrblt(RBEG[1],SBEG[1],SNUM+1);
      arrblt(RTYP[1],STYP[1],SNUM);
      RNUM←SNUM;
      SNUM←0;
      dumpout(true,crlf)


!       D2: FRC detection ;

      for I←1 thru RNUM do if RTYP[I]=NIL!
        then UFRDET(RBEG[I],RBEG[I+1])
        else
          begin
            SBEG[SNUM←SNUM+1]←RBEG[I];
            STYP[SNUM]←RTYP[I];
          end;
      SBEG[SNUM+1]←RBEG[RNUM+1];
      dumpout(true,crlf)
```

```
!        D3: correction rules ;

      RNUM←1;
      RBEG[1]←SBEG[1];
      RTYP[1]←STYP[1];
      for I←2 thru SNUM
      do begin
           if (SBEG[I+1]-SBEG[I]) ≤ 2
           then begin
                  if STYP[I]=NIL!
                  and STYP[I-1]=SIL!
                  then begin
                          STYP[I]←ASP!;    ! SIL/short NIL==>SIL/ASP ;
                        ' dumpout(true,"correction rule: "&cvs(sbeg[i]+offset)
                                  &" ASP "&cvs(sbeg[i+1]+offset)
                                  &"      SIL/short NIL —> SIL/ASP"&crlf)
                       end
                  else if (STYP[I]=NIL! and STYP[I-1]=FRC!)
                       then begin
                                  dumpout(true,("correction rule: "
                                  &segnames[styp[i-1]]&" "&cvs(sbeg[i]+offset)&" "
                                  &segnames[styp[i]]&" "&cvs(sbeg[i+1]+offset)&tab
                                  &"FRC/short NIL—>FRC"&crlf))
                                  continue;
                            end;
                 end;
                  RBEG[RNUM←RNUM+1]←SBEG[I];
                  RTYP[RNUM]←STYP[I];
          end;
       RBEG[RNUM+1]←SBEG[SNUM+1];
       SNUM←0;
       dumpout(true,crlf)


!        D4: PLS detection ;

       for I←1 thru RNUM do if RTYP[I]=NIL!
         then PLSDET(RBEG[I],RBEG[I+1])
         else
           begin
             SBEG[SNUM←SNUM+1]←RBEG[I];
             SBEG[SNUM+1]←RBEG[I+1];
             STYP[SNUM]←RTYP[I];
           end;
       dumpout(true,crlf)
```

```
!       D5: more correction rules ;

 RNUM←0;
 copy(1);
 for I←2 thru SNUM
 do
   begin
     if  STYP[I]=PLS!                            ! FRC/PLS≤2 ==> FRC ;
     and (SBEG[I+1]-SBEG[I])≤2
        then if (dum←STYP[I-1])=FRC!
         then begin
                  RBEG[RNUM+1]←SBEG[I+1];
                  dumpout(true,("correction rule: "
                          &segnames[styp[i-1]]&" "
                          &cvs(sbeg[i]+offset)&" "&segnames[styp[i]]
                          &" "&cvs(sbeg[i+1]+offset)
                          &"       FRC/short PLS-->FRC"&crlf))
              end
        else if (dum←STYP[I+1])=SIL! or dum=FRC!  ! PLS≤2/FRC or SIL==> FRC or SIL ;
          then begin
                  SBEG[I+1]←SBEG[I];
                  dumpout(true,("correction rule: "&cvs(sbeg[i]+offset)
                          &" "&segnames[styp[i]]&" "
                          &cvs(sbeg[i+1]+offset)&" "&segnames[styp[i+1]]
                          &"       short PLS/FRC or SIL-->FRC or SIL"&crlf))
               end
          else copy(I)
       else if STYP[I]=NIL!                              ! PLS/NIL≤2 ==> combine NIL to next segment ;
       and (SBEG[I+1]-SBEG[I])≤2
          then if STYP[I-1]=PLS!
           then begin
                  SBEG[I+1]←SBEG[I];
                  dumpout(true,("correction rule: "
                          &segnames[styp[i-1]]&" "&cvs(sbeg[i]+offset)
                          &" "&segnames[styp[i]]&" "
                          &cvs(sbeg[i+1]+offset)
                          &"       PLS/short NIL combine NIL to next segment"&crlf))
               end
          else copy(I)
         else copy(I)
   end;
 SNUM←0;
 dumpout(true,crlf)

!       D6: ASP detection ;

 for I←1 thru RNUM do if RTYP[I]=NIL!
   then ASPDET(RBEG[I],RBEG[I+1],(if I=1 then 0 else RTYP[I-1]))
   else
     begin
       SBEG[SNUM←SNUM+1]←RBEG[I];
       SBEG[SNUM+1]←RBEG[I+1];
       STYP[SNUM]←RTYP[I];
     end;
 arrblt(RBEG[1],SBEG[1],SNUM+1);
 arrblt(RTYP[1],STYP[1],SNUM);
 RNUM←SNUM;
 SNUM←0;
 dumpout(true,crlf)
```

```
|       D7: SP dips ;

      for I←1 thru RNUM do if RTYP[I]=NIL|
        then SPDIPS(RBEG[I],RBEG[I+1],(if I=1 then 0 else RTYP[I-1]))
        else
          begin
            SBEG[SNUM←SNUM+1]←RBEG[I];
            SBEG[SNUM+1]←RBEG[I+1];
            STYP[SNUM]←RTYP[I];
          end;
      arrblt(RBEG[1],SBEG[1],SNUM+1);
      arrblt(RTYP[1],STYP[1],SNUM);
      RNUM←SNUM;
      SNUM←0;
      dumpout(true,crlf)


        if dumpall then ! code to list pre-region spliting segmentation ;
        for I←1 thru RNUM
        do begin
             integer BT,ET,TY;
             BT←RBEG[I]; ET←RBEG[I+1];
             dumpout(true,(tab&cvs(bt+offset)&tab&segnames[rtyp[i]]
                          &tab&cvs(et+offset)&tab&cvs(et-bt)&crlf))
           end;
       ifcr notlive!version        ! code to list ZAPDASH parameters ;
        thenc
          if DUMPARS
            then
              begin
                out(CHP,tab&"DP"&tab&"DZ"&tab&"SP"&tab&"SZ");
                for I←UTTBEG thru UTTEND-1 do
                  begin
                    out(CHP,crlf&cvs(I+OFFSET)&":");
                    out(CHP,tab&cvs(PARS[I,DPDEX])&tab
                        &cvs(PARS[I,DZDEX])&tab&cvs(PARS[I,SPDEX])&tab
                        &cvs(PARS[I,SZDEX]));
                  end;
                out(CHP,crlf); if CHD≠CHP
                  then out(CHP,formfeed);
              end;
        endc ! notlive!version ;
```

```
!       D8: region splitting ;

 SNUM←0;
 for I←1 thru RNUM
 do if (dum←RTYP[I]) ≠ SIL! then REGSPLIT(RBEG[I],RBEG[I+1],dum,SNUM)
                  else begin  SBEG[add1(SNUM)]←RBEG[I];
                              STYP[SNUM]←dum;
                              SBEG[SNUM+1]←RBEG[I+1];
                       end;
 arrblt(RBEG[1],SBEG[1],SNUM+1);
 arrblt(RTYP[1],STYP[1],SNUM);
 RNUM←SNUM;


!       code to list final segmentation ;
        dumpout(true,crlf&tab&"final segmentation"&crlf);
       for I←1 thru RNUM
       do
         begin
           integer BT,ET;
           BT←RBEG[I];  ET←RBEG[I+1];
           dumpout(true,(tab&cvs(bt+offset)&tab&segnames[rtyp[I]]&tab&cvs(et+offset)
           &tab&cvs(et-bt)&crlf))
         end;
         dumpout(true,formfeed);

    end "segment";
end "segmentor";
```

6 June 1978

1. INTRODUCTION TO THE SEGMENTATION PROBLEM
2. CHOICE OF PARAMETERS
    2.1. Basis for the choice
    2.2. ZAPDASH Parameters
3. NATURE OF SOME CLASSES OF SOUNDS
4. SEGMENTATION PROCEDURE
5. PERFORMANCE EVALUATION
6. CONCLUSIONS
7. REFERENCES