Proceedings of The Institute of Acoustics

SEGLAB: AN INTERACTIVE ENVIRONMENT FOR PHONETIC SEGMENTATION AND LABELLING OF SPEECH

Art Blokland, Gordon Watson, Jonathan Dalby and Henry Thompson

Centre for Speech Technology Research
University of Edinburgh

ABSTRACT

Seglab is a component of the machine-assisted speech transcription (MAST) system under development at the Centre for Speech Technology Research, University of Edinburgh. In that system processing follows different stages: digital signal processing, segmentation and labelling, lexical lookup, and the formation of syntactic strings. Seglab constitutes the second of these stages.

The version of Seglab described here relates to a pilot system developed in the winter of 1985/86, called *RM1*. Seglab turns the acoustic quantities of signal processing into phonemic quantities. The basic technique is feature extraction, rather than template matching. The acoustic quantities are first turned into phonetic features, and then into phonemic midclasses. The midclasses are a classification of the 40-odd phonemes of English into acoustically similar groups.

Seglab is rule-based, with different kinds of rules to do the feature extraction and the construction into midclasses. It provides both an environment in which the rules are developed, and an environment in which they are run. The rules themselves are described in a companion paper [1]. The present paper describes the flow of data through the system, and the rule mechanism. It concludes with a discussion of why the system has its present form.

INTRODUCTION

The grand recognition system, from voice input to the display of text strings, falls naturally into a front end and a back end. The front end captures and digitises the signal, and produces hypotheses for the phonemic elements that the signal is supposed to contain. The back end assembles the phoneme hypotheses, first into words and then into syntactic strings. As we can expect, the hypotheses contain wrong answers as well as right ones, and the back end acts as a filter to weed out the wrong ones.

The stage at which the phonemic hypotheses are produced is Seglab, which is therefore part of the front end. The input is a set of *acoustic parameters*. The data for an acoustic parameter is an array of numbers that represent a certain measurement of the signal. The numbers are transformed into hypotheses, which are labelled segments, under the control of various kinds of rules. The labelled segments either feed directly to the next stage of processing (Lexical Access), or are written to a file for later processing.

SEGLAB: SEGMENTATION AND LABELLING

Seglab is written in Interlisp-D and runs on Xerox Dandelions. The rules that produce the hypotheses have a LISP-like syntax. Convenient facilities are provided for creating, editing, deleting and running rules. Seglab is both a development environment for these rules, and an executive for them.

The style of this paper will be to illustrate the facilities on offer by means of examples.

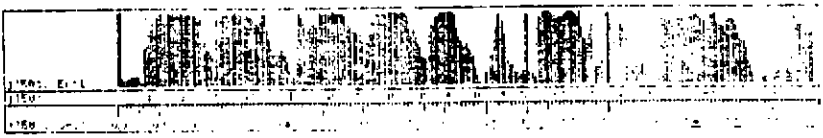
ACOUSTIC PARAMETERS

Acoustic parameters (APs) are the raw material of Seglab's input. They are the result of digital signal processing (DSP) on the digitised utterance. APs cover a range of measures, from simple channel energies computed from FFTs or LPC analysis, to formant frequency estimates. AP data is read in from files that have been prepared in advance. There is an array of data for each AP on the file. Each array contains floating-point numbers, one for each 'frame' in the utterance. A frame is an interval of time for which the DSP analysis produces a single result. A typical length for a frame is 5 milliseconds; the data for an utterance that lasts 1.7 seconds would be 340 frames long.

An example of an AP is *Erf1*, the ratio of acoustic energy in the frequency band 300-8000 Hz, against the total energy. There will be a value for Erf1 for each frame in the utterance — 340 in the above example.

Figure 1 shows a Seglab display for this parameter. Three windows are shown. At the bottom is a window with 10-millisecond tick marks, for reference. Above it is a narrow window showing a phonemic-level transcription and segmentation, produced by a phonetician and read in from a file. The transcription reads "Patty cut up a potato cake"; this was one of the test utterances on which RM1 was developed. Time-aligned with the other two windows, is the display for the AP. It consists of vertical lines, one for each frame of the utterance. There are 340 vertical lines in the diagram.

Figure 1. Seglab display of an Acoustic Parameter



Once all the APs for an utterance have been read in, they are ready to be processed. They are processed under the control of rules.

SEGLAB: SEGMENTATION AND LABELLING

THE RULE SYSTEM

The rule system is used to produce phonemic hypotheses. There are three kinds of rules: AP rules, for performing simple arithmetic on APs, T rules for producing phonetic features, and S rules for parsing sequences of features into phonemic hypotheses. S rules depend on the results produced by the other rules, and must run after them. A rule scheduler ensures that they do.

The system does not restrict what the phonemic hypotheses may be. They could be broad classes like 'stop' and 'fricative', midclasses like 'voiceless stop', fine classes like /p/, or indeed diphones, demi-syllables or transemes. For the work reported here, the hypotheses were at midclass level.

The bulk of Seglab consists of facilities to make the creation, testing and modification of these rules as painless as possible. This paper concentrates however on how the rules are supposed to be used.

AP Rules

AP rules exist to calculate new APs from APs that have been read from a file. The APs produced by the rules are called *derived APs*.

Consider again *Erf1* from the previous section. This acoustic parameter rises and falls over the utterance, depending on what was said. Suppose we are interested in the rate at which it rises and falls. Using an AP rule, we can calculate a derived AP that measures this rate. Rate-measuring derived APs are called *delta APs*, and there is a special naming convention for them. The delta of Erf1 is called *Erf1'* (read "Erf1 prime"). Running the delta rule will produce 340 values for Erf1'. Erf1 and Erf1' are both fully-fledged APs. The difference is that Erf1 is read in from a file, and Erf1' is calculated on demand, by running the appropriate AP rule.

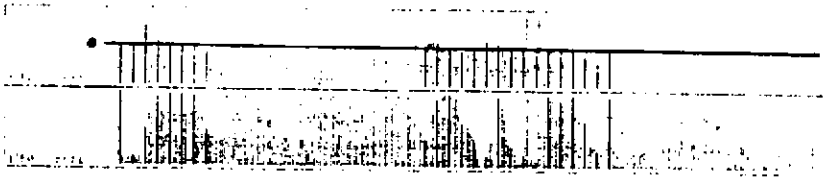
This is the text of a delta rule for Erf1:

```
Erf1':  (DIFFERENCE (Erf1 (@ Erf1 -1)))
```

`DIFFERENCE` is the standard Interlisp subtraction function, which can take two arguments. The two arguments follow in a list. The first argument is `Erf1` and the second is `(@ Erf1 -1)`. The AP rule interpreter applies the function to the arguments at each frame of the utterance; in our example it will be applied 340 times. Each time the rule is fired on a frame the value of Erf1 at the current frame and that at the previous frame are subtracted. The argument `(@ Erf1 -1)` means the value of Erf1 at the previous frame. The difference returned by the function at a frame becomes the value for Erf1' at that frame. Erf1' is both the name of the rule and the name of the derived AP that the rule produces.

Figure 2 shows the display of Erf1 again, together with that of Erf1'. When Erf1 is steady, which occurs only once in this example, Erf1' is zero.

Figure 2. Seglab display of an acoustic parameter and a derived parameter



Deltas are a typical use of AP rules, but they can be used also for calculating scaled values of APs, or the sum, difference, etc of APs.

T Rules

AP rules produce numbers from numbers. T rules take numbers to produce symbols. Suppose we want to know where the silences are in the utterance, in order eventually to identify stop closures. Usually, the acoustic energy during a stop closure falls to a low value. To test this, we run a T rule called *Silence1*, say. Here is the text for such a rule

```
Silence1:  (LESSP (Erf1 -12))
```

T rules have the same syntax as AP rules. LESSP is a standard Interlisp function, of two arguments. The rule checks whether the AP Erf1 falls below a certain threshold, in this case -12. The threshold must be provided by the author of the rule; -12dB is a reasonable value.

The operation of this T rule is as follows. At each of the 340 frames of the utterance, the function LESSP is applied, and returns *true* or *false*. If *true* is returned, the symbol *Silence1* is posted to that frame. If *false* is returned, nothing is posted. Attention then turns to the next frame. At the end of the run some of the frames have been adorned with *Silence1*, and some have not. *Silence1*, being the output of a T rule, is called a *T cat* ("threshold category"). *Silence1* is also the name of the rule.

Figure 3 shows the result of applying *Silence1*.

SEGLAB: SEGMENTATION AND LABELLING

Figure 3. Seglab display of an acoustic parameter and a threshold category (T cat)

A horizontal line has been drawn to match the frames where *Silence1* was found. The display includes the hand-label window, to show how the rule is doing. As we can see, it is not doing too badly, but it is by no means perfect.

Typically, the rule author would write several rules for silence, to capture its many acoustic characteristics. Thus *Silence2*, *Silence3*, etc, can be written, each suitable for a different context.

S Rules

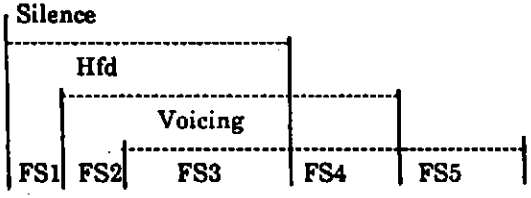
The output of a T rule is always a T cat. When a number of T rules have run, several T cats will have been established. They will have various extents, and some will overlap. Some stretches of the utterance may happen not to be labelled with T cats at all. The final stage of rule processing is to sequence, or parse, these into phonemic hypotheses. This is done by means of S rules, which are run by a Chart parser.

Active chart parsing is the framework for most of the MAST system; see [2] for details and [3] for motivation. The parser is used for different purposes at different levels in the speech chain. At the lexical level, it is used to parse phoneme sequences into words. At the syntactic level, it parses word sequences into syntactic strings. At the phonemic level here in Seglab, it parses *feature segments* into midclass hypotheses.

In traditional parsing, say words into sentences, the input constituents, namely the words, are distinct and contiguous. At the phonemic level the input constituents — let us call them phonetic features — are not contiguous. They may overlap or fall short of joining altogether. This jumble needs to be made amenable to parsing. Unlabelled gaps are repaired by a process of stretching which falls outside the scope of this paper. This leaves the overlaps. Overlapping features are an inescapable fact. The phonetic features in the present system are none other than T cats, and the T rules that produce them will never be so good that they produce unique and exhaustive labellings. To make the features amenable to parsing, we turn them into non-overlapping feature segments.

To produce non-overlapping constituents for the chart parser, we chop up the features at the feature boundaries. Figure 4 illustrates.

Figure 4. Overlapping phonetic features



A stretch of *Silence* partly overlaps a stretch of *Hfd* (high-frequency dominance), which partly overlaps a third feature, *Voicing*. A *feature segment* is a stretch in which the features are the same. The first feature segment, FS1, contains *Silence* only, and extends to the point where *Hfd* begins. *Hfd* begins a new feature segment, FS2, which contains *(Silence, Hfd)*. FS3 contains three features, *(Silence, Hfd, Voicing)*. FS4 begins where the first feature, *Silence* drops out. FS5, containing only *Voicing*, completes the fragment.

To understand how the segments are parsed, consider this S rule, for the midclass *Voiceless stop*, whose symbol here is a capital P.

```
P:   ( 10 80  +(Silence) )
     (  0 70  +(Hfd)     )
```

The voiceless stop is to be constructed, consisting of a stretch of *Silence* followed by a stretch of *Hfd*. *Hfd* represents here the aspiration after a stop release. The numbers are length constraints, in milliseconds. The stretch of *Silence*, to qualify as a constituent of a voiceless stop, must be at least 10 milliseconds long, and at most 30 milliseconds. *Hfd* must be at most 70 milliseconds long. To simplify the discussion below, we assume that these constraints are satisfied.

The parser will deliver several answers. One answer is the stretch consisting of just FS1 and FS2. FS1 contains *Silence*, as per specification. FS2, besides containing *Silence*, contains *Hfd*, also as per specification.

Another answer is the stretch consisting of FS1, FS2 and FS3. FS1 and FS2 comprise the required stretch of *Silence*, and FS3 contains the required *Hfd*. FS3 contains *Voicing* as well, but that has no bearing on the rule as specified.

A third answer is FS1, FS2, FS3 and FS4. This fullfils the specification in two ways — *Silence* from the first three segments, and *Hfd* from the fourth, or *Silence* from the first two and *Hfd* from the second two. Only one answer is delivered for these two cases, because the parser checks for redundancy.

Several more answers are produced, according to the further combinations that are possible. The final outcome is several stretches that are labelled P, all of them of different lengths.

A voiceless stop should not contain *Voicing*, and the above S rule is deficient in that respect. The feature segments FS3 and FS4, which both contain Voicing, need to be excluded. This can be done by specifying a 'minus feature' as follows:

```
P:   ( 10 80  +(Silence)  -(Voicing)  )
     (  0 70  +(Hfd)                  )
```

This rule specifies that any feature segment containing *Silence* should not also contain *Voicing*. The effect will be to exclude several possiblities from the set of answers described above.

The set of answers can be constrained still further by means of context specifications. Suppose for arguments' sake that the above is the specification for an *intervocalic* voiceless stop, and we find that the rule delivers them also for stops in other contexts. We can narrow its application by writing

```
P:   ( 10 80  +(Silence) -(Voicing)     )
     (  0 70  +(Hfd)                    )
     Left context:  (0 65535 (Voicing)
     Right context: (0 65535 (Voicing)
```

The rule will apply only if the stop is preceded and followed by a stretch of *Voicing*. The length specification (0 65535) means we do not mind how long these stretches are.

S rules are the final stage of Seglab processing. What gets passed to Lexical Access, the next link in the speech chain, is a jumble of wholly or partly overlapping phoneme hypotheses, called a phoneme lattice. There the lattice is converted into words.

DISCUSSION

Underlying Seglab is the assumption that continuous speech can be recognised automatically by means of feature extraction. Two considerations emerge from this assumption that have influenced the design.

The first is that feature extraction reduces the signal to discrete subphonemic units, which then need to be constructed into phonemic units. A language was needed in which to specify that reduction and construction, and Seglab provides it in the form of T rules and S rules. The T rules reduce acoustic material to phonetic features and the S rules construct them into phonemic hypotheses.

The second is that in using feature extraction we rely heavily on phonetic knowledge - more so than in template-matching, where the templates are usually large units like syllables or words. Given this reliance on phonetics, we needed an environment that is

congenial to phoneticians. We wanted to separate phonetics from computation, and have the phonetician deal mostly with the former. It is for this reason that Seglab is a rule-based system. Rules provide a modular way for the deployment of phonetic knowledge. They can be modified and added to at will, so that the rule base can be built up incrementally over a period of time. The typical details of computation are hidden from view, and all that is of concern are simple expressions of tests (T rules), and of sequences of phonetic features (S rules).

The success of the rule-writing enterprise is described in [1].

FUTURE DEVELOPMENT

A system consisting of threshold tests and the sequencing of features is not by itself equal to the task of recognising continuous speech, and there are several enhancements we need to make in the future. An important one is to make labelling decisions on the basis of weighted evidence rather than all-or-nothing criteria. At present, for example, we decide to label a stretch of speech as *Silence* according to the success or failure of a simple test. It would be better to consider a range of indications, and have these contribute to the decision in proportion to their importance.

REFERENCES

[1] J Dalby, J Laver and S M Hiller. 'Mid-class Phonetic Analysis for a Continuous Speech Recognition System', Proceedings of the Institute of Acoustics, Vol 8 (1986).

[2] Henry Thompson and Graeme Ritchie. 'Implementing Natural Language Parsers', in *Artificial Intelligence: Tools, Techniques and Applications*. ed Tim O'Shea and Marc Eisenstadt, Harper and Row (1984).

[3] Henry Thompson. 'Speech Transcription: An Incremental, Interactive Approach', Proceedings of the Sixth European Conference on Artificial Intelligence (ECAI-84), (1984).