THE WAL SPEECH PROGRAMMING ENVIRONMENT

P Kenne, D Landy, M O'Kane, S Nulsen, A Mitchell and S Atkins

Faculty of Information Sciences and Engineering
University of Canberra

ABSTRACT - The WAL (Wave Analysis Laboratory) Speech Programming Environment was first developed in 1987 to provide software tools for rapid prototyping of the FOPHO and SPRITE speech recognition systems. The environment has been revised and extended several times following evaluation trials at various sites. Current versions are maintained under both MS-DOS and Unix.

A central feature of the environment is a programming language which was designed to provide a high-level, natural-language-like facility for phoneticians and other speech scientists not familiar with standard programming languages to write and test speech recognition rules.

The rule language provides the usual logical operators ('and', 'or', 'not') for combining rules, together with operators for temporal reasoning ('after', 'before', 'then'). A set of primitives for describing shapes and their deformations allows the notion of 'rubber templates' to be included in the language, and when combined with the temporal reasoning facilities, provides an extension to picture languages.

The WAL environment is highly portable. The language is interpreted, with the interpreter implemented in C and the graphical user interface is implemented using X Windows. We describe the language and environment and their implementation together with a number of examples illustrating the usefulness of the WAL environment in both speech and non-speech applications.

INTRODUCTION

The origins of the Wave Analysis Language were in work designed to capture the expertise of expert (speech) waveform readers, see (O'Kane, 1983), and to provide a tool which these experts could use to write speech recognition rules without necessarily having a background in programming. To achieve these goals, the language had to provide the primitive structures which waveform readers use, together with a natural mechanism of use. The language has been in use at the University of Canberra, both for research in speech recognition and for teaching in Artificial Intelligence courses.

The Wave Analysis Language is embedded in a programming environment which contains tools which allow for easy graphical inspection of recognition rule firings (see, for example, Figure 1). There are machine learning tools which allow for automatic recognition rule parameter adjustment. The Check Tool automatically "checks" rule firings produced from the WAL interpreter working with any given set of rule files against an already-marked-up database. The tool reports current fires where the results match the database's, misfires where the results fired incorrectly, and misses where the results miss marked portions of the database. As well as reporting the starting/ending point fire/misfire/miss details for each example of each sound and word in the database the Check Tool produces a summary after each database run indicating totals for each type of rule firing.

WRITING RULES IN THE WAVE ANALYSIS LANGUAGE

Statements in WAL have an extremely simple production-rule form and are generally referred to as 'rules'. When a function in WAL is true a rule is said to have 'fired' and a label is applied. Immediate examination of the position of rule firings can be carried out either visually or aurally.

The general form of a rule is

```
feature
        name : <Rulename>,
        wave : speech,
        (association : <Condition>)
end.
```

This rule may be interpreted as "if <Condition> is true, then the rule <Rulename> holds".

The general form of a condition in a rule is

```
<Derivative> is <Feature> [with characteristic <Char>],
```

where the square brackets denote an optional construct, <Derivative> is a signal processing function, and both <Feature> and <Char> are primitive features. The set of signal processing functions and the primitive features are built in to the language. (They are both easy to extend however, with the user supplying appropriate high-level code in, for example, C or FORTRAN.)

An example of a simple rule (without the optional characteristic part) is

```
feature
        name : sloperule,
        wave : speech,
        (association : slope(w3(20000)) is
                low_amplitude(800))
end.
```

This rule holds when the slope of the w3 function (see O'Kane and Mead, 1987) calculated at a sampling rate of 20kHz is below 800. The result of applying a rule to a wave is a set of intervals (labelled with the rule name) where that rule holds. It is possible for the set of intervals to be empty.

Once a rule has been defined it may be referenced in other rules. Rules can be joned using connections. The language has the binary connectives 'and', 'or', 'then', 'before' and 'after' available (see Figure 2). The condition part of the rule may also use negation.

An example of a rule having a characteristic part is

```
feature
name : sloperule2,
wave : speech,
(association : slope(w3(20000)) is
        low_amplitude(800)
 with characteristic time_long(20))
end.
```

The segments produced by rule 'sloperule2' are those produced by rule 'sloperule' having length greater than 20 milliseconds.

The characteristics available are

time_long(Time), time_short(Time)
extend(Percentage)
extendf(Time)
extendfr(Time), extendfl(Time)

'time_long' and 'time_short' select segments of duration greater or less thanrespectively the specified time in milliseconds;

'extend' extends each end of each segment by the specified percentage; 'extendf', 'extendfr', 'extendfl' extend each segment by the specified time in milliseconds - 'extend' adds the time to each end of each segment, 'extendfr' and 'extendfl' and the time to the right and left ends of each segment respectively.

Features which are available include:

long_low_amplitude(Time,Lo)
short_low_amplitude(Time,Lo)
long_high_amplitude(Time,Hi)
short_high_amplitude(Time,Lo)
long_between_amplitude(Time,Lo,Hi)
short_between_amplitude(Time,Lo,Hi)
flat(Percentage,Number)
rise_up(Percentage1,Percentage2)
fall_down(Percentage1,Percentage2).

WAVE ANALYSIS LANGUAGE ENHANCEMENTS

Several years use of the language have proved its ease of use and it has clearly achieved its design goals. As other programming languages evolve with time, so has the WAL rule language, and we now describe the next evolutionary stage. The changes described are all based on user feedback.

A criticism of the first version of the language was that it was too verbose, requiring many keywords (e.g. feature, name, wave, etc.) per rule, and that the names of built in primitive features were also too long. Two examples of new-style rules are:

e1: fftblog(20000,100,500,220,3) is long_high_amp(10,30);

e2: fftblog(20000,100,500,220,3) is short_low_amp(10,30)
with extendf(10);

Here the rule names are e1 and e2.

A need was expressed for the ability to perform arithmetic operations on the signal processing functions. Rules of the form are now available

e3: fftband(20000,1000,2000,200,5) [+|-|*|/]
fftband(20000,3500,4000,100,2) is between_amp(10,30);

where the arithmetic is performed point-wise on the signal processing functions, and the square brackets indicate that one arithmetic operator is chosen. In addition, expressions may be more complicated:

```
2*(fftband(20000,1000,1400) + 10)/
(120 - fftband(20000,1000,1400))
```

The previous version of the language did not provide an explicit conditional expression. The form of rules involving conditionals is

```
e5:   if (male) (e4) else (e3);

e6:   if (w3(20000) is long_high_amp(10,20))
            (envtop(20000,250,3) is ext_high_amp(10,12,14))
      else
            (e2);
```

The general form of the conditional is

```
if (seg-list-object1) (seg-list-object2) [(seg-list-object3)]
```

where the seg-list-objects are expressions which evaluate to segment lists.

Composition of functions is also supported:

```
e7v2:  w3(filter(20,5),20000) is long_high_amp(10,40);
```

Other example rules are

```
e9:    slope(slope(slope(w3(20000)))) is between_amp(10,20);

e10v2:slope(w3(filter(20,4),20000)) is between_amp(10,20);

e11v2:w3(envtop(filter(20,5),20000,250,4),20000) is low_amp(4);
```

As above, the usual connectives are supported

```
e12:   e1 [and|or|then|before|after] e2;

e13:   (e1 and e2) then e3;

e15:   e1 and (not(e3));
```

The new rule language also supports a macro facility which accelerates rule development time since changing a single define will propogate throughout the rule file.

For example

```
#define  1(x,y) envtop (20000,x,y)
```

when used in a rule

```
rule :  1(250,3) is long-high-amp (10,20);
```

will expand out to

```
rule :  envtop (20,000,250,3) is long-high-amp(10,20).
```

REFERENCES

M O'Kane, "The FOPHO Speech Recognition Project", Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, pp.630-632), 1983.

M O'Kane and D Mead, "Key features in continuous speech", Proceedings of the Eleventh International Congress of Phonetic Sciences, Tallin, Vol. 3, pp.82-85, August 1987

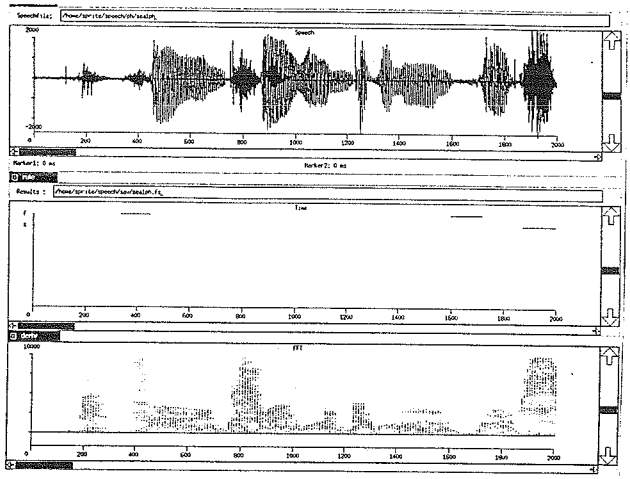

Figure 1: Example of WAL graphical rule-fire inspection and derivative display system.

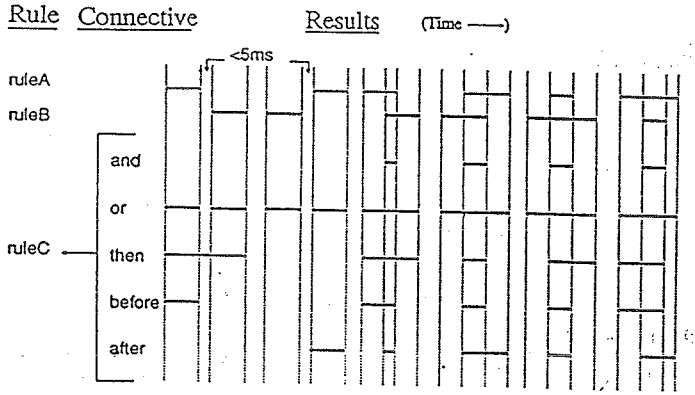

Figure 2: Chart showing rule-connective protocols