Tools and resources for speech synthesis arising from a Welsh TTS project

Briony Williams, Rhys James Jones and Ivan Uemlianin
Canolfan Bedwyr, University of Wales, Bangor, Wales, UK
Canolfan Bedwyr, Bryn Haul, Heol Victoria, Bangor LL57 2EN, Wales, UK
E-mail: b.williams@bangor.ac.uk, r.j.jones@bangor.ac.uk, i.uemlianin@bangor.ac.uk

Abstract

The WISPR project ("Welsh and Irish Speech Processing Resources") has been building text-to-speech synthesis systems for Welsh and for Irish, as well as building links between the developers and potential users of the software. The Welsh half of the project has encountered various challenges, in the areas of the tokenisation of input text, the formatting of letter-to-sound rules, and the implementation of the "greedy algorithm" for text selection. The solutions to these challenges have resulted in various tools which may be of use to other developers using Festival for TTS for other languages. These resources are made freely available.

1. Introduction

The Welsh-language researchers in the WISPR project ("Welsh and Irish Speech Processing Resources") have developed diphone-based and limited-domain Welsh voices for the Festival speech synthesis system (details on Festival can be found at http://www.festvox.org). In the course of this work, certain Welsh-specific and other challenges were encountered, some of which are described in this paper. These include the areas of tokenisation of input text, formatting of letter-to-sound rules, and implementing the "greedy algorithm" for optimal text selection for use in unit selection TTS. In the course of meeting these challenges, various tools have been developed, which may be of use to other developers using Festival. These are made freely available.

2. Tokenisation of input

2.1 Background

In a text-to-speech (TTS) system, tokenisation is the process whereby an input stream of text is divided up into units suitable for further processing. The output will be in a form suitable for further processing by the lexicon or the low-level synthesis routines.

User feedback was received after the restricted release of a beta version of the TTS system. It was reported that long Welsh texts were not correctly processed by the system. This was mainly due to the lack of tokenisation rules, leading to a failure to handle many forms of punctuation, and any tokens that were not dictionary words. The only text processing carried out by the existing system involved the inclusion of a few acronyms, abbreviations and numerals in the lexicon. Hence the rapid development of a tokenisation module was a priority.

In addition, there was a need for our system to handle input text in UTF-8 format, which includes the common accented characters (vowels with circumflex, grave and acute accents) as well as w-circumflex and y-circumflex characters, which are fairly common in Welsh but not in other languages. Since Festival seems unable to handle these characters, it was necessary to write new text-handling code as part of the Festival software.

After examining on-line texts and analysing user requirements, a list of tokenisation priorities was drawn up: punctuation; ordinal and cardinal numbers; currency amounts; time phrases; common acronyms; percentages and keyboard modifiers (e.g. Ctrl+S), for use with screen readers. Many of these categories are dealt with similarly as in English, but as will be described, numbers (ordinals and cardinals) and time phrases present particular challenges in Celtic languages.

2.2 Ordinal and cardinal numbers

2.2.1 Numbering systems

In common with other Celtic languages (Kvale and Foldvik, 1997), Welsh has two systems for numbering quantities over ten (Roberts, 2000), as follows. In current Welsh education, the accepted method of counting is the newer decimal system, which expresses 97 as "naw deg saith" (literally "nine ten(s) seven"). The more traditional method of counting in Welsh is based on a partly vigesimal (base-20) system, which also contains elements of base-15 counting. Using this system, the number 97 would be tokenised as "dau ar bymtheg a phedwar ugain" (two on fifteen, and four twenties).

For cardinal numbers up to one US billion (a thousand million), the decimal system is used in the tokenisation system.

To use the decimal system for low values of ordinal numbers would strike a native speaker as unnatural. Therefore, the vigesimal system is used for ordinals up to 100. Beyond this, the current practice of pre-pending the cardinal number with "rhif" ("number") is used. The term "97ain" is thus tokenised as "ail ar bymtheg a phedwar ugain", but "197ed" as "rhif cant naw deg saith".

2.2.2 Consonant mutation and numbering

Welsh, in common with other Celtic languages, displays the phenomenon of consonant mutation. Under this system, certain syntactic, morphological and lexical conditions trigger a phonetically determined change in the initial consonant of words.

The numeric tokenisation process needs to be aware of mutation effects. For example, 900 would be transcribed as “naw cant” (“nine hundred”), but 200 should be uttered as “dau gant”, as nouns following “dau” (“two”) require a soft mutation. Multiples of thousands are affected in a similar way. The recursive process used to tokenise numbers takes account of the mutated forms and uses them when required.

2.2.3 Time phrases

Time phrases are detected by the tokenisation module if they are in the form a:b (if 0<=a<=23 and 0<=b<=59). In Welsh, the vigesimal system is used in time phrases.

The algorithm developed also correctly tokenises o’clock, quarter to/past and half past. For Welsh, it therefore allows 11:42 to be tokenised as “deunaw munud i ddeuddeg” (literally, “two-nines minutes to twelve”, i.e. “eighteen minutes to twelve”), and 11:45 to become “chwarter i ddeuddeg” (“quarter to twelve”).

2.3 Handling UTF-8 input

It seemed that Festival was unable to handle UTF-8 text completely, whether as text in a file of input, or when used in interactive command-line mode. Hence it was decided to use the following strategy for enabling Festival to handle input text in UTF-8 format:

- Write new software within Festival (in the C language) that converts Welsh UTF-8 characters to equivalent strings in a 7-bit format (e.g. “a+” for “â”, or “a/” for “á”).
- Retain the existing LTS rules and lexicon, which use text in 7-bit format only.

The resulting C code was merged into the main Festival code. The patch can be downloaded from:
http://bedwyr-redhat.bangor.ac.uk/svn/repos/WISPR/Software/Festival/WISPR/Patch/Trunk/festival_utf8.patch
Full details of the input characters handled is at http://bedwyr-redhat.bangor.ac.uk/svn/repos/WISPR/Software/Festival/WISPR/Merged/Trunk/festival/src/modules/Text/text_welsh.cc

2.4 Tools for general use

The following resources have been made available:

- The UTF-8 software, which can be easily adapted to use other subsets of Unicode, for other languages.
- The tokenisation software for Welsh, which can be adapted for use with other languages, including those using a vigesimal counting system.

3. Letter-to-sound rules

In a speech synthesis system, the letter-to-sound (LTS) rules carry out the mapping from the input orthography to the output string of phonemes, for all words that have not first been found in the lexicon.

3.1 Background

In Welsh, the correspondence between orthography and pronunciation is very close, and so manually written letter-to-sound rules are feasible. A set of rules had been written by hand for an older TTS system (Williams 1992, 1993, 1994). These took the form of three sets of rules (corresponding to three passes through the input) to carry out: epenthetic vowel insertion, stress location, and the grapheme-to-phoneme conversion proper. It was proposed to use these rules rather than spend time producing a new set. Although the older Welsh TTS system had been ported to Festival in the past , there were many errors in the newly-ported LTS rules (for instance, the functionality of “zero or more of” etc. had been completely lost). However, the original rules were written in a different format from that used in the Festival TTS system, being a relic of the older C-based programming method that had been used. Hence there was a need to convert them to the “Scheme” format used in Festival.

It was felt that the conversion process would yield a tool that would be potentially useful to rule-writers for other languages where manually-written rules were appropriate. The Scheme-based format is more difficult for human rule-writers to use, since the spaces between every symbol make it a little unclear what the rules are doing. However, the more “linguist-friendly” format of the older rules makes them easy for a user to read, write and debug. Hence it was felt to be worthwhile to provide a means whereby developers could write rules in the “linguist-friendly” format and then automatically convert them (accurately) into the Festival format.

3.2 The task

The existing rules were in the form of critically-ordered context-sensitive rewrite rules, in the following format:
ng[w]H=M
In this string, the target character “w” is rewritten as “M” when preceded by the symbols “ng” and followed by one or more of “a,e,i,o,u,y” (denoted by the variable “H”). This rule precedes a more general rule which rewrites all remaining cases of input “w” to “w”. The rules are able to use variables (such as “H” above), and the variables can be specified in terms of the following:

- One and only one of (some set of symbols);
- One or more of;
- Zero or more of;

These “logical conditions” on the variables were an important aspect of many rules, and the lack of them (in the first attempt at porting the system to Festival) had led to errors in the rule output. So it was clear that any conversion process would ned to retain this functionality.

3.3 The solution

A Python script (“lff2scm.py”) was written to convert from the older format to the Festival format (see Uemlianin 2005a). This script expects the input rules to have the following format:

To begin with, there is an optional list of output symbols, with interpretation. Next follows a list of any variables. These are each specified in terms of:

- One and only one (the default).
- Zero or one (i.e. optional).
- Zero or more.
- One or more.

Then follow the rules themselves, with comments (which will be passed to the Scheme file as comments).

The Python script was found to be very convenient for porting LTS rules to the Festival Scheme format. The result was that the new TTS system had far more accurate LTS rules than did the earlier attempt at porting the Welsh synthesiser to Festival.

3.4 A tool for general use

Although Festival offers the option of training a statistical method for determining pronunciation, this can be a long and difficult process in the case of a language that lacks a digital pronunciation lexicon. Therefore it is preferable to use manual LTS rules in the following situation:

- The correspondence between orthography and pronunciation is close enough to make manual rules feasible and reasonably accurate.
- There is no existing digital pronunciation lexicon for the language

It is hoped that the “lff2scm” script may be useful to other workers who are writing manual LTS rules for languages where these two conditions apply. It is freely downloadable (Uemlianin 2005a), and the Python software is also freely available from www.python.org.

4. Optimal text selection

4.1 The task

When developing a unit-selection-based text-to-speech (TTS) system (e.g., using the Festival speech synthesis system), the first steps are to design and collect a corpus of naturalistic speech, i.e., of spoken text. This corpus should contain all (or as near as possible) of the required units (usually diphones) of the target language.

Designing such a corpus is a significant task. A common shortcut is to collect a large amount of text and select a subset of this text which maximises the unit coverage, and ideally minimises the amount of text to be read. The standard algorithmic approach is the greedy set-cover algorithm (Cormen et al., 2001).

Our solution to this problem was to develop Optese (Uemlianin, 2005). We found only one software package on the internet applicable to this problem, namely OTS (“Optimal Text Selection”) from the Local Language Speech Technology Initiative, LLSTI (Bali et al., 2004). Optese maximises coverage and minimises text significantly more efficiently than OTS.

4.2 Comparison

4.2.1 Availability and documentation

OTS is available from the LLSTI tools download page (http://www.llsti.org/downloads-tools.htm). The package includes usage documentation and an FAQ. Optese is available from its webpage (see Uemlianin, 2005b), which includes guidance on installation and usage, and information on the program's limitations.

4.2.2 Performance

The test data used was a list of 9321 Welsh and English sentences, in orthographic and phonemic representations. The table below shows time taken, units found and sentences needed for those units, for each tool.

Tool	Time taken	Units Found	Sentences Needed	Units per Sentence
OTS	12 hours	10101	1767	5.72
Optese	5 mins	9268	1996	4.64

Table 1: Comparative evaluation of Optese and OTS

Although OTS shows a higher units per sentence ratio, it should be noted that:

- Optese was significantly faster (5 minutes as against 720 minutes, i.e. 0.7% of the time taken by OTS).
- Many of the units OTS collected were actually double-counted, due to the faulty implementation of the regular-expression-based parsing in OTS (e.g., occurrences of the diphone @-n were also counted as occurrences of the diphone @@-n). Consequently, the OTS figures are not reliable.

Given the extremely large time penalty involved with OTS, it was felt that Optese was the more practical tool to use, yielding results that were at least as good if not better.

4.2.3 Design

OTS and Optese implement the same algorithm. OTS is written in C++ (a fast compiled language), Optese is written in Python (a relatively slow interpreted language). So the question is: why is Optese so much faster?

The main reason for this lies in the relative complexity of the data structures which the two programs use in implementing the algorithm, as follows:

- Optese uses the simplest possible data structures, namely sets: each sentence, and each selection of sentences, is represented as a set of phones. Sets are simpler than lists, as sets do not include ordering information or duplicate elements.
- OTS, on the other hand, builds a matrix of diphones: a sentence for example being represented as a path through the matrix. The memory footprint and the processing for OTS are correspondingly heavy, and increase exponentially with the size of the phoneset.

The matrix representation in OTS answers questions Optese cannot. For example, given a unit X, OTS could determine the probability of a unit Y occurring within an arbitrary range. OTS could perhaps serve as the basis of an n-gram language model generator.

4.3 A further implementation

After we had conducted this study, we discovered that the unofficial festvox-2.1 distribution (downloadable from http://festvox.org/latest/festvox-2.1-current.tar.gz) contains a set of scripts (in src/promptselect/) with similar functionality to Optese. There is a brief comment in the scripts themselves, and two slides in a CMU-internal lecture (see http://festvox.org/festtut/slides/lecture13.pdf, slides 21 & 22.). But apart from these, the routine "promptselect" has no documentation at all. The scripts are hardcoded to require a specific (US English) phoneset and lexicon, so they would not be appropriate for processing Welsh data (or any other language). Optese, on the other hand, is language-independent. In addition, it is better documented than "promptselect".

4.4 Frequency information

Optese collects frequency information for the units it finds. This allows the user to specify the number of examples of each unit to collect, and a minimum frequency threshold (e.g., "only collect units which occur more than 10 times in the data"). However, more could be made of this information. For example, when designing a minimal corpus for speaker adaptation in ASR, the aim is to collect data whose phonemes have a specific frequency distribution (e.g., Cui and Alwan 2002; Nagórski et al 2003). Expanding Optese to provide this functionality would be feasible and appropriate.

4.5 A tool for general use

In designing Optese we have focussed on usability and fitness-for-purpose. Clear documentation and fluid performance are important objectives. Also, we have aimed to create a tool that is of use to other researchers. It is freely available for download (Uemlianin 2005b).

5. Conclusion

In the course of a project developing TTS for Welsh, tools have been developed which may be of use to researchers working on Festival-based TTS. All these resources are available for download over the web. The tools include:

- lff2scm: a Python script to convert LTS rules from an easy-to-read format to the Festival Scheme format.
- Tokenisation code for Welsh which can be adapted to other languages, including those with a vigesimal counting system.
- Software that reads UTF-8 input and converts it into 7-bit format for processing by Festival.
- Optese, a new implementation of the "greedy algorithm" for optimal text selection for unit selection synthesis, which is significantly faster and better documented than two other such algorithms.

6. Acknowledgements

The WISPR project was funded by INTERREG,, an initiative of the European Union to facilitate co-operation between adjacent EU regions. Additional funding was provided by the Welsh Language Board.

7. References

Bali, K., Talukdar, P.P., Krishna, N.S. and Ramakishnan, A.G., (2004). Tools for the Development of a Hindi Speech Synthesis System, 5th ISCA Speech Synthesis Workshop. www.llsti.org/pubs/hpl_isca_paper.pdf

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2001) Introduction to Algorithms. MIT Press.

Cui, X. and Alwan, A. (2002). Efficient Adaptation Text Design Based on the Kullback-Leibler Measure. Proceedings of IEEE ICASSP 2002.

Kvale, K. and Foldvik, A.K., (1997) Four-and-twenty, twenty four. What's in a number? In Proceedings of Eurospeech 1997 (pp. 729--732).

Nagórski, A., Boves, L., and Steeneken, H. (2003). In search of Optimal Data Selection for Training of Automatic Speech Recognition Systems. Proc IEEE Automatic Speech Recognition and Understanding Workshop 2003, St. Thomas, US Virgin Islands.

Roberts, G.. (2000) Welsh Number Talk. In Proceedings of the Sixth Annual Conference of the North American Association for Celtic Language Teachers (NAACLT). See http://www.naaclt.org

Uemlianin, I.A. (2005a) lff2scm (information/download). http://www.bangor.ac.uk/~cbs007/lff2scm/lff2scm.html

Uemlianin, I. A. (2005b) Optese: Optimal Text Selection. www.bangor.ac.uk/~cbs007/optese/README.html

Williams, B (1992) Welsh letter-to-sound rules for text-to-speech synthesis. In: Proceedings of the Institute of Acoustics, vol. 14.

Williams, B. (1993) Letter-to-sound rules for the Welsh Language. In: Proceedings of the Third European Conference on Speech Communication and Technology (Eurospeech 93), Berlin, Germany.

Williams, B. (1994) Welsh letter-to-sound rules: Rewrite rules and two-level rules compared. Computer Speech and Language, vol. 8: pp. 261--277.