Speech Synthesis in Festival

A practical course on making computers talk

Edition 1.4.1, for Festival Version 2.0

last updated May 8th 2000

by Alan W Black Language Technologies Institute, Carnegie Mellon University
(with content from Paul Taylor (Edinburgh) and Michael Macon (OGI))

NOTE: this document is incomplete

This document was generated on 30 May 2000 using the texi2html translator version 1.52.

Go to the first, previous, next, last section, table of contents.

---

1 Introduction

The purpose of this course is

- To allow understanding of the basic parts of speech synthesis
- To understand the relative complexity of implementing solutions to the problems
- To become familiar with the Festival architecture and know what it can and can't do

As a firm believer in learning by doing this course tries to touch on every aspect of speech synthesis from a practical view. General discussion of problems are dicussed, with some presentation of potential theoretical solutions. Where appropriate, substatial exercises are given which will hopefully lead to greater understanding of the actual problems.

In addition to discussion, exercises will be given with hints about how to do them in Festival. The exercises may take some time and are sometimes open ended with no obviously right solution. This reflects the synthesis field.

1.1 Text to Speech

Many people have quite different views of what processes are involved in text to speech. Often what are considered primary areas by some are considered trivial by others. This is partly due to different problems in different languages but also due to researchers only seeing the part that interests them (a common problem in most areas of research). Here we also present a particular view of the processes involved in text to speech, which is probably also biased but does at least discuss all those parts that are necessary for our system to run.

In this course we will view TTS in four major parts

1. Architecture: something that can hold the system together. This introduces clearly defined objects and the processes that we wish to apply to them.
2. Text processing: Analysis of raw and labelled text into identifiable words. This covers tokenization, mapping tokens to words, resolving homographs, and explicit mark-up languages.
3. Linguistic/prosodic processing: From words to segments, F0 and durations (and anything else appropriate for waveform synthesis). This deals with lexicons for pronunciation of words, intonation (prosodic phrase, accent and F0 prediction) and durations.
4. Waveform synthesis: From segments, F0 and duration to a waveform. There are many techniques

to do this, concatenative synthesis (diphone, unit selection), formant synthesis and articulatory synthesis.

In addition to these sub-areas of speech synthesis there are also general aspects which touch on all of these aspects and are best dealt with as separate parts.

1. Building new voices: either in existing supported languages or new languages
2. Building domain specific voices: by tailoring modes, (text analysis, lexicon, prosody) or by recording specific databases and building limited domain synthesizers.
3. Building data-driven models of speech: for prosody, letter to sound rules, and text analysers.
4. How to use speech synthesis: what it can do and what it can't. How to get the best possible synthetic voice for different applications
5. Concept-to-speech vs Text-to-speech

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

2 History

Here is a (probably biased and naive) history of generating synthetic speech.

The idea that a machine could generate speech has been with us for some time but the realization of such machines has only really been practical in the last 50 years, only in the last 20 years have we actually seen what could be termed practical examples of text to speech systems.

First you must be aware of some distinctions here. The creation of synthetic speech covers a whole range of processes and although often all lumped under the general term *text-to-speech* much work has concentrated on how to generate the speech itself from known phones rather than the whole text convertion process.

Probably the first practical application of speech synthesis was in 1936 when the UK Telephone Company introduced a speaking clock. It used optical storage for the phrases, words and part-words which were appropriately concatenated to form complete sentences.

Also around that time Homer Dudley at Bell Labs, developed a mechanical device that through movement of pedals, and mechanical keys, like an organ, with a trained operator could generate almost recognizable speech. Called the *Voder* it was demonstrated at the 1939 World's fair in New York and San Francisco. A recording of this device exists, can be heard as part of a collection of historical synthesis examples that were distributed on a record as part of *klatt87*. See http://www.festvox.org/history/klatt.html.

The realization that the speech signal could be decomposed as a source filter model was utilized to build analogue systems that could be used to mimic human speech. The *vocoder*, also developed by Homer Dudley is one such example. Much of the work in synthesis in the 40s and 50s was primarily concerned with constructing the signal itself rather than generating the phones from some higher form like text.

Further decomposition of the speech signal allowed the development of *formant synthesis* where collections of signals were composed to form recognization speech. Predicting the parameters that adequately represent the signal has always (and still is) difficult. Early versions of formant synthesis allowed these to be specified by hand but automatic prediction was the goal. Today formant synthesizer can produce high quality recognizable speech if the parameters are properly adjusted. But it is still difficult to get full natural sounding speech from them when the process is fully automatic.

With the advancement of digital computers the move to digital representations of the speech and reduction of a dependence of specialized hardware, more work was done in concatenation of natural recorded speech. It quickly became ovbious that *diphones* were the desired unit of presentation. That is, units of speech from middle of one phone to the middle of another were the basic building blocks. The justification is that phone boundaries are much more dynamic than middle of phones and therefore mid-phone is a better place to concatenate that phone boundary. The rise of concatenative speech starting in the 70s until today has to a large extent been practical because of the reduction in the cost of eletronic storage. When a megabyte of memory was a signiciant part of researchers salary, less resource intensive techniques were naturally investigated more. Of course formant synthesis often requires significant computational power, even if it requires less storage, so even into the 80s speech synthesis relied on specialized hardware.

In 1972 the standard Unix manual (3rd edition) included commands to process text to speech, form text analysis, prosodic prediction, phoneme generation and waveform synthesis through a specialized piece of hardware. Of course Unix had only about 16 installations at the time and most (all?) were located in Bell Labs at Murray Hill.

Techniques were being developed to compress speech in a way that it could be more easily used in applications. The Texas Instraments *Speak 'n Spell* toy, released in the late 70s, was one of the earlier examples of mass production of speech synthesis. The quality was pretty poor, but for the time it was very impressive. Speech was basically encoded using LPC (linear Predictive Coding) and mostly used isolated words and letters though there were also a few phrases formed by concatenation. Simple TTS engines based on specialised chips became popular on home computers such as the BBC Micro in the UK and the Apple][.

Dennis Klatt's MITalk synthesizer *allen87* in many senses defined the perception of automatic speech synthesis to the world at large. Later developed into the product DECTalk, its produces a somewhat robotic, but very understandable form of speech. It is a formant synthesizer (reflecting its development period).

Before 1980 speech synthesis research was limited to large laboratories that could afford to invest the time and money for hardware. By the mid-80s more labs and universities started to join in as the cost of the hardware dropped. By the late eighties purely software synthesizers became feasible, that not only produced reasonable quality speech, but also could do so in near real-time.

Of course with faster machines and large disk space people began to look to improving synthesis by using larger, and more varied inventories for concatenative speech. Yoshinori Sagisaka at ATR in Japan developed nuu-talk *nuutalk92* in the late 80s early 90s, which used a much larger inventories of concatenative units thus instead of one example of each diphone unit there could be many and a automatic acoustic based selection was used to find the best selection of sub-word units from a fairly geenral database of speech. This work was done in Japanese which has a much simpler phonetic structure than English making it possible to get high quality with still a relatively small databases. Though even by

94 the generation of the parameter files for a new voice in nuu-talk (503 senetences) would take several days of CPU time, and synthesis was not generally real-time.

With the demonstration of general *unit selection synthesis* in English in Rob Donovan's PhD word *donovan95* and ATR's CHATR system (*campbell96* and *hunt96*) by the end of the 90's unit selection became the hot topic for speech synthesis research. However inspite of very high quality examples of it working, generalized unit selection also produces some very bad quality synthesis. As the optimial search and selection agorithms used are not 100% reliable both high and low quality synthesis is produced.

Of course the development of speech synthesis is not in isolation from other developments in speech technology. With the success in speech recognition, also benefiting from the reduction in cost of computational power and increased availability of general computing into the populace. There are now many more people who have the computational resouces and interest in running speech applications. This ability to run such applications puts the demand on the technology to deliver both working recognition and acceptable quality speech synthesis.

Availability of free and semi-free synthesis systems such as the MBROLA project and the Festival Speech Synthesis System makes the cost of entering the field of speech synthesis much lower and many more groups have now joined in the development.

However although we are now at the stage were talking computers are with us there is still much to do. We can now build synthesizers of (probably) any language that produced reconizable speech. But if we are to use speech to receive information as easily as we do from humans there is still much to do. Synthesized speech must be natural, controllable and efficient (both in rendering and in the building of new voices).

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

12 References

allen87
J. Allen, S. Hunnicut, and D. Klatt. *Text-to-speech: The MITalk system*. Cambridge University Press, Cambridge, UK., 1987.

anderson84
M. Anderson, J. Pierrehumbert, and M. Liberman. Synthesis by rule of English intonation patterns. In *Proceedings of ICASSP 84*, pages 2.8.1--2.8.4, 1984.

bacchiani96
M. Bacchiani, M. Ostendorf, Y. Sagisaka, and K. Paliwal. Design of a speech recognition system based on acoustically derived segmental units. In *ICASSP-96*, volume 1, pages 443--446, Atlanta, Georgia, 1996.

bachenko90
J. Bachenko and E. Fitzpatrick. A computational grammar of discourse-neutral prosodic phrasing in English. *Computational Linguistics*, 16(3):155--170, 1990.

black96
A. Black and A. Hunt. Generating F$_0$ contours from ToBI labels using linear regression. In *ICSLP96*, volume 3, pages 1385--1388, Philadelphia, PA., 1996.

black98b
A. Black, K. Lenzo, and V. Pagel. Issues in building general letter to sound rules. In *Proc. ESCA Workshop on Speech Synthesis*, pages 77--80, Australia., 1998.

black97b
A. Black and P. Taylor. Assigning phrase breaks from part-of-speech sequences. In *Eurospeech97*, volume 2, pages 995--998, Rhodes, Greece, 1997.

black97a
A. W. Black. Predicting the intonation of discourse segments from examples in dialogue speech. In Y. Sagisaka, N. Campbell, and N. Higuchi, editors, *Computing Prosody*, pages 117--128. Springer-Verlag, 1997.

black95a
A. W. Black. Comparison of algorithms for predicting accent placement in english speech synthesis. In *Proceedings of the Acoustics Society of Japan*, pages 275--276, 3--4--1, Spring, 1995.

black95d
A. W. Black and N. Campbell. Optimising selection of units from speech databases for concatenative synthesis. In *Eurospeech95*, volume 1, pages 581--584, Madrid, Spain, 1995.

campbell96
N. Campbell and A. Black. Prosody and the selection of source units for concatenative synthesis. In J. van Santen, R. Sproat, J. Olive, and J. Hirschberg, editors, *Progress in speech synthesis*,

pages 279--282. Springer Verlag, 1996.

campbell91

N. Campbell and S. Isard. Segment durations in a syllable frame. *Journal of Phonetics*, 19(1):37--47, 1991.

campbell92b

W. N. Campbell. Synthesis units for natural English speech. *IEICE*, SP 91-129:55--62, 1992.

conkie96

A. Conkie and S. Isard. Optimal coupling of diphones. In J. van Santen, R. Sproat, J. Olive, and J. Hirschberg, editors, *Progress in speech synthesis*, pages 293--305. Springer Verlag, 1996.

DeRose88

S. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31--39, 1988.

donovan95

R. Donovan and P. Woodland. Improvements in an HMM-based speech synthesiser. In *Eurospeech95*, volume 1, pages 573--576, Madrid, Spain, 1995.

dusterhoff97a

K. Dusterhoff and Black A. Generating F$_0$ contours for speech synthesis using the tilt intonation theory. In *Proc. ESCA Workshop on Intonation,* Athens, Greece., 1997.

dutoit93

T. Dutoit and H. Leich. MBR-PSOLA : Text-to-speech synthesis based on an MBE re-synthesis of the segments database. *Speech Communication,* 13:435--440, 1993.

fujimura93

O Fujimura. C/d model: a computational model of phonetic implementation. In E Ristad, editor, *DIMACS Proceedings*. Am. Math. Soc., 1993.

fujisaki83

H. Fujisaki. Dynamic characteristics of voice fundamental frequency in speech and singing. In P MacNeilage, editor, *The Production of Speech*, pages 39--55. Springer-verlag, 1983.

hess83

W. Hess. *Pitch Detection in Speech Signals: Algorithms and Devices*. Springer Verlag, 1983.

hirschberg92

J. Hirschberg. Using discourse content to guide pitch accent decisions in synthetic speech. In G. Bailly and C. Benoit, editors, *Talking Machines*, pages 367--376. North-Holland, 1992.

hirschberg94

J. Hirschberg and P. Prieto. Training intonation phrase rules automatically for English and Spanish text-to-speech. In *Proc. ESCA Workshop on Speech Synthesis*, pages 159--162, Mohonk, NY., 1994.

huang97

X. Huang, A. Acero, H. Hon, Y. Ju, J Liu, S. Meredith, and M. Plumpe. Recent improvements on microsoft's trainable text-to-speech synthesizer: Whistler. In *ICASSP-97,* volume II, pages 959--962, Munich, Germany, 1997.

hunt96

A. Hunt and A. Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *ICASSP-96*, volume 1, pages 373--376, Atlanta, Georgia, 1996.

hunt89

M. Hunt, Zwierynski D., and Carr R. Issues in high quality LPC analysis and synthesis. In *Eurospeech89*, volume 2, pages 348--351, Paris, France, 1989.

jilka96

M. Jilka. Regelbasierte generierung nat\"urlich klingender intonationsmuster des amerikanischen englisch (rule-based generation of naturally sounding intonation patterns of american english). Master's thesis, University of Stuttgart, Institute of Natural Language Processing, 1996.

kain98

A. Kain and M. Macon. Spectral voice conversion for text-to-speech synthesis. In *ICASSP-98*, volume 1, pages 285--288, Seattle, Washington, 1998.

klatt87

D. Klatt. Review of text-to-speech conversion for english. *Journal of the Acoustical Society of America*, 82:737--793, 1987.

malfrere97

F. Malfrere and T. Dutoit. High quality speech synthesis for phonetic speech segmentation. In *Eurospeech97*, pages 2631--2634, Rhodes, Greece, 1997.

malfrere98

F. Malfrere, T. Dutoit, and P. Mertens. Automatic prosody generation using suprasegmental unit selection. In *Proc. ESCA Workshop on Speech Synthesis*, pages 323--327, Australia., 1998.

marcus93

M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313--330, 1993.

moebius96

B. Moebius. Synthesizing german intonation contours. In J.P. van Santen, R. Sproat, J. Olive, and J. Hirschberg, editors, *Progress in Speech Synthesis*, pages 401--415. Springer, 1996.

moehler98

G. Moehler and A. Conkie. Parametric modelling of intonation using vector quantization. In *Proc. ESCA Workshop on Speech Synthesis*, pages 311--316, Australia., 1998.

moulines90

Eric. Moulines and F. Charpentier. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech Communication*, 9(5/6):453--467, 1990.

ostendorf95

M. Ostendorf, P. Price, and S. Shattuck-Hufnagel. The Boston University Radio News Corpus. Technical Report ECS-95-001, Electrical, Computer and Systems Engineering Department, Boston University, Boston, MA, 1995.

ostendorf94

M. Ostendorf and N. Veilleux. A hierarchical stochastic model for automatic prediction of prosodic boundary location. *Computational Linguistics*, 20(1):27--55, 1994.

pierrehumbert80

Janet B. Pierrehumbert. *The Phonology and Phonetics of English Intonation*. PhD thesis, MIT, 1980. Published by Indiana University Linguistics Club.

ritchie92

G. Ritchie, G. Russell, A. Black, and S. Pulman. *Computational Morphology*. MIT Press,

Cambrdige, Mass., 1992.

ross96

K. Ross and M. Ostendorf. Prediction of abstract prosodic labels for speech synthesis. *Computer, Speech and Language*, ??(?):?, 1996.

nuutalk92

Y. Sagisaka, N. Kaiki, N. Iwahashi, and K. Mimura. ATR -- ν-TALK speech synthesis system. In *Proceedings of ICSLP 92*, volume 1, pages 483--486, 1992.

sanders95

E. Sanders and P. Taylor. Using statistical models to predict phrase boundaries for speech synthesis. In *Eurospeech95*, volume 2, pages 1811--1814, Madrid, Spain, 1995.

silverman92

K. Silverman, M. Beckman, J. Pitrelli, M. Ostendorf, C. Wightman, P. Price, J. Pierrehumbert, and J. Hirschberg. ToBI: a standard for labelling English prosody. In *Proceedings of ICSLP92*, volume 2, pages 867--870, 1992.

sproat98b

R. Sproat, A. Hunt, M. Ostendorf, P. Taylor, A. Black, K. Lenzo, and M. Edgington. SABLE: A standard for TTS markup. In *International Conference on Spoken Language Processing*, Sydney, Australia, 1998.

sproat96b

R. Sproat, C. Shih, W. Gale, and N. Chang. A stochastic finite-state word-segementation algorithm for Chinese. *Computational Linguistics*, 22(3), 1996.

sproat97

R. Sproat, P. Taylor, M. Tanenblatt, and A. Isard. A markup language for text-to-speech synthesis. In *Eurospeech97*, volume 2, pages 995--998, Rhodes, Greece, 1997.

syrdal98a

A. Syrdal, G. Moehler, K. Dusterhoff, A. Conkie, and Black A. Three methods of intonation modelling. In *Proc. ESCA Workshop on Speech Synthesis*, pages 305--310, Australia., 1998.

taylor00a

P. Taylor. Analysis and synthesis of intonation using the tilt model. *Journal of the Acoustical Society of America*, 107 3:1697--1714, 2000.

taylor99b

P. Taylor and A. Black. Speech synthesis by phonological structure matching. In *Eurospeech99*, volume 4, pages 1531--1534, Budapest, Hungary, 1999.

taylor98b

P. Taylor, A. Black, and R. Caley. The architecture of the festival speech synthesis system. In *3rd ESCA Workshop on Speech Synthesis*, pages 147--141, Jenolan Caves, Australia., 1998.

taylor94b

P. Taylor and A. W. Black. Synthesizing conversational intonation from a linguistically rich input. In *Proc. ESCA Workshop on Speech Synthesis*, pages 175--178, Mohonk, NY., 1994.

taylor97b

P. Taylor and A. Isard. SSML: A speech synthesis markup language. *Speech Communication*, 21:123--133, 1997.

bosch98

A. van den Bosch, T. Weijters, and W. Daelemans. Modularity in inductive-learned word pronunciation systems. In *Proc. NeMLaP3/CoNNL98*, pages 185--194, Sydney, 1998.

yarowsky96

D. Yarowsky. Homograph disambiguation in text-to-speech synthesis. In J. van Santen, R. Sproat, J. Olive, and J. Hirschberg, editors, *Progress in speech synthesis*, pages 157--172. Springer Verlag, 1996.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

3 Festival Overview

Festival has been built to address a number of specific issues in speech synthesis research and development. The biggest problem with speech synthesis development is that in order to improve one part you need all the other parts of the system before you can test your new development. Festival is specifically designed to provide an environment where you may develop your own small part and use the other already written modules without having to start from scratch. It also allows you to test multiple theories in exactly the same environment, something that is important in evaluating research.

3.1 Users of Festival

There are three specific types of users we are aiming at:

- multi-lingual text to speech: for those who have little interest in the internal workings of the system, and just want speech output.
- Synthesis for language system: for applications that generate text from known forms. In this type of system perhaps telephone numbers, addresses, etc. can be explicitly marked, language type, even intonational forms can be specified. This form of access requires more knowledge about the synthesis internals but still not its low level details.
- Synthesis development environment: In this mode, new synthesis modules, intonation, waveform synthesizers, etc. can be developed and compared in a software environment that provides the right basic tools so that development may concentrate on the theory not the implementation.

Importantly what makes this approach worthwhile is that because development happens within the same basic system as applications actually use. there is a direct route from research to use.

The above three schemes may be mixed and many people will use Festival at different levels. Each level roughly corresponds to a different programming language. In the multi-lingual tts mode work would most probably be done in a Unix shell or other API, integrating Festival into some larger application. In language mode, parameters would be specified in Scheme, Festival's scripting language. In developing new synthesis method, work would most likely be done in C++.

3.2 Core system

The core system consists of the following features.

- Scheme-based scripting language: In order for easy specification of parameters and flow of control within the system a Scheme interpreter (a dialect of Lisp) is provided as a command interpreter. This means much of Festival's features are fully controllable at run time without having to re-compile the system.
- C++/C core modules: The core modules are written in C++ or C and are easily interfaced to the Scheme

interpreter. This offers both the advantages of a fast efficient language and the flexibility of an interpreted system.
- General utterance representation: In C++ classes are used to offer a flexible powerful representation for utterances. This makes writing functions using utterances easy and efficient. This is provided by the Edinburgh Speech Tools Library
- Waveform I/O, formats, resampling: Many common waveform formats are cleanly supported so that waveforms, label files, coefficient files can easily be read and written. Resampling and changing formats is also supported making portability much easier.
- Utterance, relations, features, I/O: The utterance structure gives a common regular form to all utterances. Full support for access through Scheme, and in C++ is made through simple to use functions. Utterances, or parts of utterances may be dumped to files in a human readable form for external manipulation and reloaded.
- Standard data tools: A number of basic tools are available so you can easily use standard methods without having to build new tools. These include a Viterbi decoder, ngram support, regular expression (Regex) matching, linear regression support, CART support (though the CART builder `wagon'), weighted finite state transducers, and stochastic context free grammars.
- Audio device access and spooling: The Edinburgh Speech Tools Library offers direct and indirect support for many types of output audio device. Also *spooling* is supported, allowing synthesis to continue while playing a file.
- server/client model: A server client mode is provided for so that a larger more powerful machine might be used remotely by smaller programs saving on both start up time, and resources required on the client end.

3.3 Festival 1.4.1

The current version of Festival offers the following key features.

- English, (British and American), and Spanish text to speech
- Externally configurable language independent modules: phonesets, lexicons, intonation, part of speech, duration, diphone/unit selection, letter-to-sound rules, text modes.
- On-line documentation: HTML and info. Also meta-h in the command interpreter will give help on the current symbol.
- Example applications: saytime, latest news
- Portable (Unix) distribution, (preliminary Windows NT/95 support)
- Multiple APIs: STML, emacs, scripting, shell, client/server.

3.4 Festival uses

Festival offers a number of APIs for synthesis.

- Unix shell:

```
festival --tts news.txt
echo "Hello world" | festival --tts
```

- Emacs: Say menu: say region, buffer, select language, select voice
- Interactive command interpreter Scheme based read-eval-print loop
- C++ library adding modules in C++

- client/server mode

3.5 Using Festival

This is the recommended method for using Festival as part of this course. One of the main objectives of this course is that at least you'll be able to make your computer talk using Festival, saying the things you wish. Of course it is also intended that you understand the processes involved in generating that speech and that you understand enough that you can influence these methods.

The documentation is on-line in GNU Emacs info mode and in HTML format at

```
http://www.speech.cs.cmu.edu/festival/manual-1.4.1
```

See the "Quick Start" chapter in the user manual.

Because much of this course involves writing, or more usually copying and modifying, small pieces of Scheme code and rules, the following mode of working is recommended.

Add any new rules, functions, parameter settings to a file and always name that file as an argument to Festival when you start it. For example create a file called `` `ex.scm' `` in some new directory. Add the following to it

```
;;; Functions. rules, parameters etc for festival course
;;;
(Parameter.set 'Duration_Stretch 2.0)
```

Start Festival as follows

```
festival ex.scm
```

Now when you synthesize anything it should be very slow

```
(SayText "This is a pen")
```

Remember to remove the `Duration_Stretch` command before doing the other exercises.

3.6 Exercises

These exercises are designed to work with Festival version 1.4.1. Some auxiliary programs and files are given as part of the course they will identified with respect to the `COURSEDIR` which will be installed on the system where you will be doing these exercises. Ask the course organiser for the actual pathname.

1. Make Festival say your name, (adding an entry to the lexicon if your name is not pronounced correctly).
2. Make Festival say the names of all people logged onto liddell (or some large central machine).
3. Install the Emacs interface. Select a piece of text in a buffer and get Festival to say it. Find ten things that

Festival doesn't say properly. (we will try to fix those things later in the course).
4. How long does it take for Festival to say "Alice's Adventures in Wonderland"?

3.7 Hints

This section gives hints (and sometimes the full answers) to the exercises. There is a lot to learn in starting to use Festival so these hints are here to point you in the right direction. Also we provide things like shell scripts etc. that are not part of Festival itself, but help to complete the exercises.

1. This can be done from the shell

   ```
   echo My name is ... | festival --tts
   ```

   or within the command interpreter with the command

   ```
   (SayText "My name is ...")
   ```

   in the command interpreter. If your name is not pronounced properly you can add new entries to the lexicon using the the function `lex.add.entry` For example the default synthesizer pronounces Ronald Reagan's second name wrongly so we can redefine the pronunciation as

   ```
   (lex.add.entry
    '("reagan" n (((r ei) 1) ((g a n) 0))))
   ```

   To find out what the phoneme set is and possible formats, it is often useful to lookup similar words. Use the `lex.lookup` function as in

   ```
   (lex.lookup 'reagan)
   ```

   then copy the entry changing it as desired. To keep the pronunciation add it to your `` `.festivalrc' `` in your home directory. This file is automatically loaded every time you run Festival so then it will always know about your name. Because there are different lexicons for different languages/dialects you must first select the lexicon/voice first before setting the new pronunciation.

   ```
   (voice_kal_diphone)
   (lex.add.entry ...)
   ```

2. You'll need to get the list of names of the people who are logged on. You can do this in a number of different ways. One way is

   ```
   #!/bin/sh
   #
   who | awk '{print $1}' | sort -u |
   while read i
   do
      cat /etc/passwd | grep "^"$i":" | awk -F: '{print $5}' | sed 's/,.*$//'
   ```

```
done
```

This program is also in `COURSEDIR/bin/users-logged-on'. You may want to add to this list with some introduction like "The people currently logged on to liddell are:" and you may need to add lexical entries for some names.

3. Read the chapter on the Emacs interface in the Festival manual. Change your `.emacs' accordingly. `festival.el' is already installed in an directory accessible to Emacs in CSTR.
4. To save you typing the whole book, an on-line copy of Lewis Carroll's story is in

```
ftp://uiarchive.cso.uiuc.edu/pub/etext/gutenberg/etext91/alice30.txt
```

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

4 Architecture

4.1 Overview

In any large systems there has to be an explicit notion of *architecture*. The basic objects, and processes of the system have to be explicitly identified. One of the major problems within almost any reserach field is that the researchers do not have experience in large system buildinbg and often spend more time on the reserach than on system management leading to a very unweildly system which is difficult to modify and maintain.

In the MITalk book, *allen87* a basic *pipeline architecture* is used. Basically each module reads in some structured information form the preceeding module and outputs a new structure for the succeeding module. This model has the disadvantage that anything that is needed by a later module must be preoperly outputed by each intervening module. This is especially difficult when the modules are different (e.g. Unix) processes as in the Bell Labs 1972 system.

An alternative architecture is a *blackboard architecture* where a global is accessible to all modules which may add and modify as these require. The exact line between such architectures is fuzzy

4.2 Utterance architectures

Orthogonal to the issue of pipeline versus blackboard there is the important aspect of how the respentation of an utterance itself should be structure. Choosing the right structure is very important. As it defines what can and cannot be done easily within the archtecture.

@cindex{string model} The first such utterance structure is the *string model*. Here a string of synbols is incrementally modified through each modules. For example if we start with a string of tokens.

```
We started on Feb 25.
```

The first expansion modules would replace all tokens with words to give a string

```
We started on february twenty fifth .
```

Other modules would replace words with phones, then phoens with phones+durations etc. In this model we have effectively unstructured data sitting in a simple list at each stage.

@cindex{multi-level data structures} Another problem that is not catered for in the simplest string model is that information about previosu levels is lost at each stage. Thus *multi-level data structures* can be used so that each

new module adds new levels, without removing old ones.

This again can be done in multiple ways. One way is to build a table where bounaries denote times in the eventual synthesized utterance. Thus we have something like

```
| Feb                           | 25                                        |
| february                      | twenty                  | fifth           |
|    1       |   0    | 0  | 0  |         1      |     0  |     1           |
| f | eh | b | r | ax | er | iy | t | w | eh | n | t | iy | f | ih | f | th |
```

Thus giving layers for tokens, words, syllables and phones.

Unfortunately this nice heirarchical structure doesn't always match what you want to do. Intonation accents and boundaries can best be done orthogonal to syllables. Diphones also cross over boundaries. A second rpoblem is that although we now have multi-levels each level is still a simple list. IF we wish to add syntactic parsing, or prosodic phrasing tree represenations would be best. This ouwl require further additions to this structure. The third aspect to consider is traversal of this tructure to find related information. Thus what is the mechanism to find the first phone in a word. Although all such "problems" can be dealt with in this model some are stretching the structure in ways that it was not originally designed to be used.

It is not unusual that the basic structure set up to hold complex objects in a system, although carefully designed at the start, becomes a burden such that new ideas and concepts become so difficult to implement within that structure that such enhancements are not considered.

4.3 Festival utterance structure

The fundamental object used in Festival system is the *utterance*. Each module in the system is given an utterance which it will manipulate in some way and then pass on to the next module. Given Festival started as a new system recently it is able to benefit from the limitations found in previous systems.

4.3.1 Utterances, relations and items

An *utterance* consists a set of *items* which are related through a set of *relations*. An *item* may be in one or more relations. Each relation consists of a list or tree of items.

Items are used to represent objects like words or segments, but also sometimes more abstract objects like a node in a syntax tree. An *item* has a number of *features* associated with it. Each feature has a feature name and feature value, the name is a simple striing while the value may be a string, integer or float, (or also any other complex object).

Relations are used to link *items* together is useful ways. Such as in a list of words, or a syntax tree or the syllable structure. Individual items may be in multiple relations, for example a word will be in the `Word` relation as well as being the `SylStructure` relation as the root of a tree describing its syllable structure.

For example a basic utterance can contain a number or relations. Here we show a `Word`, `Syllable` and `Segment` list relations with a tree-structured `SylStructure` relation overlayed.

We call this structure a *heterogeneous relation graph* (or HRG). A more detailed dicussions of the benefits and its relationship to other structures is given in *taylor98b*.

4.3.2 Standard relations

There is no fixed set of relations and new ones can easily be added, at run-time, or existsing ones ignored. However in the standard English voice a set of standard relations are used and wil be refered to though this course.

Text
: a character string of the utterance.

Token
: a list of trees. This is first formed as a list of tokens found in a character text string. Each root's daughters are the *Word*'s that the token is related to.

Word
: a list of words. These items will also appear as daughters (leaf nodes) of the `Token` relation. They may also appear in the `Syntax` relation (as leafs) if the parser is used. They will also be leafs of the `Phrase` relation.

Phrase
: a list of trees. This is a list of phrase roots whose daughters are the `Word's` within those phrases.

Syntax
: a single tree. This, if the probabilistic parser is called, is a syntactic binary branching tree over the members of the `Word` relation.

SylStructure
: a list of trees. This links the `Word`, `Syllable` and `Segment` relations. Each `Word` is the root of a tree whose immediate daughters are its syllables and their daughters in turn as its segments.

Syllable
: a list of syllables. Each member will also be in a the `SylStructure` relation. In that relation its parent will be the word it is in and its daughters will be the segments that are in it. Syllables are also in the `Intonation` relation giving links to their related intonation events.

Segment
: a list of segments (phones). Each member (except silences) will be leaf nodes in the `SylStructure` relation. These may also be in the `Target` relation linking them to F0 target points.

IntEvent
: a list of intonation events (accents and bounaries). These are related to syllables through the `Intonation` relation as leafs on that relation. Thus their parent in the `Intonation` relation is the syllable these events are attached to.

Intonation
: a list of trees relating syllables to intonation events. Roots of the trees in `Intonation` are `Syllables` and their daughters are `IntEvents`.

Wave
: a single item with a feature called `wave` whose value is the generated waveform.

4.3.3 Items and features

To access information in items in an utterance a simple feature mechanism has been implemented. Each item holds features named by a string, feature values may be strings, integers or floats. For example here is how to access an utterance through features in Scheme. Suppose we create an utterance as follows

```
(set! utt1 (SayText "The book is on the table"))
```

We can extract the first word from this

```
(set! firstword (utt.relation.first utt1 'Word))
```

We can find the part of speech of this word by accessing its `pos` feature.

```
(item.feat firstword "pos")
```

As well as basic features other complex feature names allow access to other parts of the utterance. For example to find the first segment in this word

```
(item.feat firstword "R:SylStructure.daughter1.daughter1.name")
```

Dot separated tokens in the feature name may refer to other items related to the given item either within the current relation or through others. A number of direction operators are defined `n` (next), `p` (previous), `daughter1` (first daughter), `daughter2` (second daughter), `daughtern` (last daughter), `parent`, `first` (most previous), `last` (most next). Also the token immediately following the prefix `R:` is treated as a relation name and current relation is switched to it. For example

```
(item.feat firstword "n.pos")
```

Accesses the next word's part of speech.

```
(item.feat firstword "n.R:SylStructure.daughter1.stress")
```

Accesses the next words, first syllable's stress value.

If a feature name doesn't point to a valid place (e.g. there is no next item. `"0"` is returned.

Features allow a uniform method for accessing the utterance allowing many simple models, such as CART trees, linear regression etc. to take parameters in a clean way. For example a decision tree to assign accents on stressed syllables in content words or on unstressed syllables in single-syllable content words may be represented as

```
((R:SylStructure.parent.gpos is content)
 ((stress is 1)
  ((Accented))
```

```
  ((position_type is single)
   ((Accented))
   ((NONE))))
 ((NONE))))
```

where `R:SylStructure.parent.gpos`, `stress`, and `position_type` are all features.

4.4 Synthesis modules

4.4.1 Utterance types and modules

Each *utterance* also has a *type*. Synthesis is defined in terms of the type of an utterance.

A *module* is a process that can be applied to an *utterance*.

For example the following creates an utterance of type `Text`

```
(Utterance Text "hello")
```

When synthesized, using the `utt.synth` function the utterance's type, `Text`, defines which modules get run on the input.

```
(defUttType Text
  (Initialize utt)
  (Text utt)
  (Token utt)
  (POS utt)
  (Phrasify utt)
  (Word utt)
  (Intonation utt)
  (Duration utt)
  (Int_Targets utt)
  (Wave_Synth utt))
```

Or when segments are explicitly included in the input none of the higher level analysis need be done

```
(Utterance Segment ((h 0.058) (@ 0.039)
                    (l 0.069) (ou 0.219)))

(defUttType Segment
  (Initialize utt)
  (Wave_Synth utt))
```

Choices between different modules of the same type (e.g. different duration modules) are done through the

`Parameter` mechanism. For example if you want to select `Klatt` duration rules use

```
(Parameter.set 'Duration_Method 'Klatt)
```

Or if you wish to select average duration then use

```
(Parameter.set 'Duration_Method 'Averages)
```

To find out the appropriate parameter names and values consult the relevant chapters in the manual.

4.5 Exercises

1. Copy `` `text2pos' `` and modify it to output the number of nouns (of any type) in a given file.
2. Copy `` `text2pos' `` and modify it to output the number of vowels (phoneme vowels not letter vowels) in a given file.
3. Using the sentence "Mr. Rogers moved to Pittsburgh on 25 May 1976." create a Text utterance and hand synthesize it by applying each of the modules. At each stage identify which relations are created and what new features are added to the items.

4.6 Hints

1. ```
   (set! total_ns (+ 1 total_ns))
   (format t "Total number of nouns %d\n" total_ns)
   ```

2. See `` `SPPPDIR/src/festival/lib/synthesis.scm' `` for the definition of `Tokens` UttType for list of extra modules to call. You want to look at the Segment relation

   ```
   (if (string-equals (item.feat seg "ph_vc") "+")
       (set! total_vs (+ 1 total_vs))
   )
   ```

3. See `` `\$SPPPDIR/src/festival/lib/synthesis.scm' `` for the definition of `Text` UttType for list of modules to call. The following functions are useful:

   ```
   (utt.relationnames utt)
   (utt.relation.items utt 'Segment)
   ```

   The following actually works for trees and lists

   ```
   (utt.relation_tree utt 'Segment)
   (set! seg4 (nth 4 (utt.relation.items utt 'Segment)))
       (item.features seg4)
   ```

   ```

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

5 Text processing

The first of the three major tasks in speech synthesis is the analysis of raw text into something that can be precessed in a more reasonable manner.

In this section we will look at how to take arbitrary text and convert it to identifiable words chunked into reasonable sized utterances.

5.1 Text analysis

Consider the following examples to see how directly the written form follows standard pronunciation.

- This is a pen.
- My cat who lives dangerously has nine lives.
- He stole $100 from the bank.
- He stole 1996 cattle on 25 Nov 1996.
- He stole $100 million from the bank.
- It's 13 St. Andrews St. near the bank.
- My home pgae is http://www.cstr.ed.ac.uk/~awb.

Or worse still consider this mail message (even with the headers deleted).

```
from awb@cstr.ed.ac.uk ("Alan W Black") on Thu 23 Nov 15:30:45:
>
>  ...  but, *I* wont make it :-) Can you tell me who's going?
>
  IMHO I think you should go, but I think the followign are going
     George Bush
     Bill Clinton
     and that other guy

Bob

--

 +-------------------------------------------------+  |\\   //|
 | Bob Beck  E-mail bob@beck.demon.co.uk           |  | \\ // |
 +-------------------------------------------------+  |  > <  |
                                                      | // \\ |
```

```
                                        Alba gu brath |//___\\|
```

In the above there are a number of specific problems a speech synthesizer needs to address before it could adequately rendered this message as speech. At least the following need attention

- The quoted part needs to have its ">" symbols removed.
- Emphasized word needs to be recognized
- Smiley, IMHO
- Spelling error in "followign"
- Identify there is a list of names
- Identify ascii art.
- Identify foreign language.

We can split the task down as follows

- identify tokens in the text
- chunk the tokens into reasonably sized sections
- identify types for tokens (some pronounced, some not)
- map tokens to words
- identify types for words

5.2 Identifying tokens

Whitespace (space, tab, newline, and carriage return) can be viewed as separators.

Punctuation can also be separated from the raw tokens.

Festival converts text from files into an ordered list of tokens each with its own preceding whitespace and succeeding punctuation as features of the token.

5.3 Chunking into utterances

"Sentences end with a full stop." Unfortunately it is nowhere near as simple as that. We wish to chunk the text into reasonable sized chunks so that they can synthesized quickly and played so that there is as little time as possible between utterances, especially at the start.

Most synthesizers support some form of *spooling* which allows the next utterance to be synthesized while the previous is actually playing. Synthesis time varies from machine to machine and synthesizer to synthesizer. There can be many factors and they are not all due to the actual algorithms used in the synthesizer. Resampling time, pauses introduced by audio hardware, and accessing files on remote disks often play as much in the overall timing as the algorithms and the machine speed itself.

In Festival we chunk tokens into utterances which are what can be most reasonably recognized as sentences.

Ideally chunks should be prosodic phrases but that would require more analysis of the tokens before a decision about where the prosodic phrases should occur can be made. Festival uses a rule system to determine utterance breaks, it depends on the token itself and its context. It allows lookahead to the next token.

The currently used decision tree for determining end of utterance is as follows

```
((n.whitespace matches ".*\n.*\n[ \n]*") ;; A significant break in the text
  ((1))
  ((punc in ("?" ":" "!"))
   ((1))
   ((punc is ".")
    ;; This is to distinguish abbreviations vs periods
    ;; These are heuristics
    ((name matches "\\(.*\\..*\\|[A-Z][A-Za-z]?[A-Za-z]?\\|etc\\)")
     ((n.whitespace is " ")
      ((0))                 ;; if abbrev signal space is enough for break
      ((n.name matches "[A-Z].*")
       ((1))
       ((0))))
     ((n.whitespace is " ")  ;; if it doesn't look like an abbreviation
      ((n.name matches "[A-Z].*")  ;; single space and non-cap is no break
       ((1))
       ((0)))
      ((1))))
    ((0)))))
```

Thus the above difficult cases try to deal with the case where a token is terminated by a period but could be an abbreviation. An abbreviation is recognized as containing a dot or capitalized with one or two letters or three capital letters. When an abbreviation is detected there must be more than one space and the next word must be capitalized to signal a break. If the word doesn't appear to be an abbreviation, then any long break or capitalized following word will signal a break.

This will fail for such examples as

- cog. sci. Newsletter.
- many cases at end of line.
- Badly spaced/capitalized sentences.

5.4 Tokens to words

Now we have our chunks we can relate each token to zero or more words. This requires context-sensitive rules. This takes into account, numbers, dates, money, some abbreviations, email addresses, random tokens. This is to identify and analyze tokens that have some internal structure. Note that some have different readings depending on dialect even when we agree on what they represent.

```
1100 meters -> eleven hundred or one thousand one hundred
$3.50 -> three dollars (and) fifty (cents)
```

Pronunciation of numbers often depends on its type.

- 1776 date: seventeen seventy six.
- 1776 phone number: one seven seven six.
- 1776 quantifier: one thousand seven hundred (and) seventy six
- 25 day: twenty fifth

An example rule for dealing with such phrases as "$1.2 million" would be

```
(define (token_to_words token name)
 (cond
  ((and (string-matches name "\\$[0-9,]+\\(\\.[0-9]+\\)?")
        (string-matches (item.feat token "n.name") ".*illion.?"))
   (append
    (builtin_english_token_to_words token (string-after name "$"))
    (list (item.feat token "n.name"))))
  ((and (string-matches (item.feat token "p.name")
                        "\\$[0-9,]+\\(\\.[0-9]+\\)?")
        (string-matches name ".*illion.?"))
   (list "dollars"))
  (t
   (builtin_english_token_to_words token name))))
```

But even this isn't 100% robust.

@cindex{homographs} Homographs are words that are written the same but are pronounced differently. There are a number of different types of homographs which can be distinguished through different types of information.

- Different part of speech: `project`.
- Semantic difference: `bass, tear`.
- Proper names: `Nice, Begin, Said`.
- Roman Numerals: `Chapter II, James II`.
- Numbers: years, days, quantifiers, phone numbers.
- Some symbols: `5-3, high/low, usr/local`.

Care should be taken when trying to deal with a phenomena. The question is, 'How often does it occur?'. The answer depends very much on the type text you are dealing with.

- Numbers: email 2.57%, novels (Conan Doyle stories) 0.00013%.
- Part of speech homographs: Wall Street Journal 7.6%.

There are however relatively few semantic homographs in languages, English has perhaps only a few hundred, this is probably related to the fact that it would be too difficult to read if too many words were homographs.

Although some phenomena can only realistically be treated by hand written rules, others depend on a host of different contextual information and can be better learned from suitable data. Festival supports a homograph disambiguation method based on *yarowsky96*. This technique covers all classes of homographs, those based on part of speech as well as "semantic" ones (e.g. "row" (boat and argument)), though part of speech homographs are usually dealt with by the part of speech tagger.

To use this disambiguation technique, first you need a *large* corpus of words. We've been using a collection of over 63 million words including, various novels, newspapers, encyclopedia, research papers and email.

The stages involved in building a disambiguator for a particular homograph are as follows.

1. Extract all occurrences of homograph from corpus.
2. Label each occurrence with its class
3. Extract contextual features that will identify class
4. Build classification tree (or decision list) to classify occurrences.

This may seem like an impossibly large task but it is surprising how few occurrences of homographs there actually are. For example there are only 442 occurrences of the token `bass` in our corpus, and 6167 occurrences of the word `lead`.

Of course for some homographs there are *many* more. For examples there are hundreds of thousands of tokens containing only digits in our corpus. In this case we simply take some representative subset.

The features used to classify are relatively easy to find and improve on. Content words in the context of five words before and five word after are very good at disambiguating many semantic tokens, as discussed in `yarowsky96`. The classification tree for the token `St` for street or saint, is dependent of the capitalization and punctuation of the immediately surround words. It is often that while labelling the class of the occurrences of homographs, potential discriminatory features come to mind.

Using this technique we build a classifier for tokens containing only digits. Because there are many occurrences of such tokens we only used examples from two sub-corpora, namely four years of Time Magazine and 10 years of personal email. We classified around 100,000 occurrences of numbers into four classes, `years`, `days` (pronounced as ordinals), `quantifiers` and `phone numbers` (pronounced as digits). The distribution in the corpus was 42% numbers, 35% years, 19% days, and 3% phone numbers. We achieved an overall 97.4% correct classification on held out test data.

This technique is successful *if* the data has the appropriate coverage. But due to the changing nature of language we find that no matter how big your training set is, there still will be apparently common forms which never appear in it. For example, in building disambiguators for roman numerals, our databases has no occurences of the words `Pentium` or `Palm` appearing before roman numerals even though in today's text these are common.

5.5 Summary of Festival's text processing

When Festival performs TTS what it really does is

- Tokenize the text into an ordered list of tokens
- Chunk the tokens into utterances
- Apply user defined functions to each utterance, typically this is `utt.synth` and `utt.play`
- The function `utt.synth` runs further analysis on each token in an utterance converting it to one or more words.
- Text modes allow a special filter for the whole file and the specification of mode specific parameters such as token-to-word functions.

5.6 Text modes

Another level of control that is desirable is mode specific processing, even when we have no explicit mark-up. Obviously the treatment of some tokens will be different between different types of text. For example a "/" (slash) inside a token in an email message is much more likely to identify a Unix path name than when it appears within a Reuters news article, where it probably identifies alternatives. Email messages have conventions for quoting (and signatures) which can be dealt with that make comprehension of the message much easier. Also some file formats have partial mark-up that is useful. Latex has methods for marking emphasis which can easily be detected.

Festival supports the notion of *text modes*, following the notion of modes in Emacs, which allow customization of mode specific parameters.

Specifically it offers

filter
: A Unix program filter for the file. In email-mode this removes most of the mail headers.

init_function
: A Scheme function to be called when entering the mode. This allows selection of voice, addition of lexical entries, and mode specific tokenization rules to be set up.

exit_function
: Called on exiting the mode, so you can tidy everything up and not leave mode specific rules that cause other synthesis modes to fail.

5.7 An example email text mode

In this example we will show a text mode which deals with email messages. It is not complete but shows some of the things you can do. We will filter the message extracting interesting parts (sender, subject and body). For token to word rules, we will set rules for email addresses, and remove greater than signs in quoted paragraphs. We will also switch voices for quoted text.

First we define a filter that extracts the from line and subject from the headers and the message body itself

```
#!/bin/sh
#  Email filter for Festival tts mode
#  usage: email_filter mail_message >tidied_mail_message
grep "^From: " $1
echo
grep "^Subject: " $1
echo
# delete headers (up to first blank line)
sed '1,/^$/ d' $1
```

Now we define the init and exit functions. In this small example the only thing we do is save the existing text to token function and cause this mode to use ours. In the exit function we switch things back.

```
(define (email_init_func)
 "Called on starting email text mode."
 (set! email_previous_t2w_func token_to_words)
 (set! english_token_to_words email_token_to_words)
 (set! token_to_words email_token_to_words))

(define (email_exit_func)
 "Called on exit email text mode."
 (set! english_token_to_words email_previous_t2w_func)
 (set! token_to_words email_previous_t2w_func))
```

The function `email_token_to_words` must be defined. We'll discuss it in three parts.

```
(define (email_token_to_words token name)
  "Email specific token to word rules."
  (cond
   ((string-matches name "<.*@.*>")
     (append
      (email_previous_t2w_func token
       (string-after (string-before name "@") "<"))
      (cons
       "at"
       (email_previous_t2w_func token
        (string-before (string-after name "@") ">")))))))
```

This function will be called for each token in an utterance. It is called with two arguments: the item (`token`) and the actual token's string are given (`name`).

The first clause identifies an email address and removes the angle brackets then calls the token-to-word function recursively to say the name and address separately.

The second part of this function is designed to identify quotes in an email message.

```
   ((and (string-matches name ">")
         (string-matches (item.feat token "whitespace")
                         "[ \t\n]*\n *"))
    (voice_don_diphone)
    nil ;; return nothing to say
   )
```

That is when the token is a *greater than* sign and it appears at the start of a line. When this is true we select the alternate speaker and return no words to be said (i.e. the greater than sign is silent). This also has the advantage that the word relation that will be created in the utterance will be continuous over the newline and quote marker so it wont interfere with prosodic phrasing.

The third part deals with all other cases and simply calls the previously defined token to word function. But before that we must find out if we have switched back to unquoted text and hence need to switch back the speaker.

```
   (t  ;; for all other cases
     (if (string-matches (item.feat token "whitespace")
                         ".*\n[ \n]*")
         (voice_gsw_diphone))
     (email_previous_t2w_func utt token name))))
```

Now the next stage is to define the mode itself. This is done through the variable `tts_text_modes` like this

```
(set! tts_text_modes
   (cons
    (list
      'email   ;; mode name
      (list         ;; email mode params
       (list 'init_func email_init_func)
       (list 'exit_func email_exit_func)
       '(filter "email_filter")))
    tts_text_modes))
```

You can test this mode with the example mail message in `` `FESTIVALDIR/examples/ex1.email' ``. You must load all of the above commands into Festival before the email mode will work. Save them in a file and name that file as an argument when starting Festival, then call tts on the file like this

```
(tts "FESTIVALDIR/examples/ex1.email" 'email)
```

Note there a number of specific problems. No end of utterance is detected after "URL." and before "Alan" in the quoted text. This is because our end of utterance rules (as described above) don't deal with this case. Can you

see how to modify them to fix this?

Other problems too exist, such as end of quoted text may not be detected if there is no blankline between the quoted and unquoted forms.

There is the question of whether text modes should produce STML or not. If they did produce STML then they could easily port to other synthesizers.

5.8 Mark-up languages

But wouldn't it all be easier if the input included information identifying what the text was? In many uses of synthesis such information does exist in the application and could easily be passed on to the synthesizer if there was a well defined way to do so. Synthesis uses such as language generation, machine translation, dialog systems and information providing systems often know significant details about what has to be said.

Sable is an XML-based language developed for marking up text (*sproat98b*, which is based on STML *sproat97*, and on previous work on SSML *taylor97b*). Sable was designed by a group invovling AT&T, Bell Labs, Sun, Apple, Edinburgh University and CMU. It is intended as a standard that can be used across many different synthesis systems.

Input in Sable may be labelled, identifying pronunciation, breaks, emphasis etc.

An example best illustrates its use.

```
<?xml version="1.0"?>
<!DOCTYPE SABLE PUBLIC "-//SABLE//DTD SABLE speech mark up//EN"
        "Sable.v0_2.dtd"
[]>
<SABLE>
<SPEAKER NAME="male1">

The boy saw the girl in the park <BREAK/> with the telescope.
The boy saw the girl <BREAK/> in the park with the telescope.

Some English first and then some Spanish.
<LANGUAGE ID="SPANISH">Hola amigos.</LANGUAGE>
<LANGUAGE ID="NEPALI">Namaste</LANGUAGE>

Good morning <BREAK /> My name is Stuart, which is spelled
<RATE SPEED="-40%">
<SAYAS MODE="literal">stuart</SAYAS> </RATE>
though some people pronounce it
<PRON SUB="stoo art">stuart</PRON>.  My telephone number
is <SAYAS MODE="literal">2787</SAYAS>.
```

```
I used to work in <PRON SUB="Buckloo">Buccleuch</PRON> Place,
but no one can pronounce that.

By the way, my telephone number is actually
<AUDIO SRC="http://www.cstr.ed.ac.uk/~awb/sounds/touchtone.2.au"/>
<AUDIO SRC="http://www.cstr.ed.ac.uk/~awb/sounds/touchtone.7.au"/>
<AUDIO SRC="http://www.cstr.ed.ac.uk/~awb/sounds/touchtone.8.au"/>
<AUDIO SRC="http://www.cstr.ed.ac.uk/~awb/sounds/touchtone.7.au"/>.
</SPEAKER>
</SABLE>
```

Sable currently supports

- selection of language, voice, genre.
- marking of boundaries and emphasis.
- definition of pronunciation for words.
- pronunciation of literals (spelling) and phonemes.
- Specification of intonation and speaking rate
- inclusion of sound files.

But Sable is still being developed and other tags are under consideration including synthesis engine specific commands and labelling of phrase types (e.g. "question", "greeting" etc.)

There is much interest in defining such a mark-up up language which is independent of any particular synthesizer and work is continuing with a number of important laboratories of agreeing on a standard.

5.9 Normalization of Non-Standard Words

A cleaner alternative general method for text normalization. This model was developed as a part of a project at the 1999 Summer Workshop at Johns Hopkins University. See http://www.clsp.jhu.edu/ws99/projects/normal/ for full description.

The idea behind this model was to formalize the notion of text analysis so that there need be only a small number of clearly defined, and hopefully trainable, modules that offer translation from strings of characters to lists of words.

In the *NSW* framework there are 4 basic stages of processing:

splitter
: A simple tokenizer, that splits not only "classic" whitespace separated tokens but also within such tokens where punctuation (e.g. hyphens) or capitalization suggest such a split.

type identifier
: for each split token identify its type, one of around 20 types, identifying how the toke is to be expanded.

token expander
: for each typed toke expand it to words. In all except one case this expnation is pretty much deterministic, such as number, date, money, letter sequences expansion. Only in the case of abbreviations is some extra required.

language modelling
: A langauge model is then used to select between possible alternative pronucniations of the output.

This project wanted to look at how a text normalization function could be trained for different domains. To do this four basic text types were chosen.

NANTC
: (North American News Text Corpus), presswire database from, New York Times, Wall Street Journal etc. This was chosen as a baseline that many text conditioners and TTS engines are already tuned for. This consists of about 4.3 million tokens, of which around 8.8% were considered non-standard words

Classified Ads
: Because this genre is so productive we collected 415K tokens from various websites of real estate classified ads. 43.4% of these were non-standard words.

PC110
: To allow testing on data similar to email (and data that could be freely distributed), we extracted 264K tokens from a mailing list of the IBM PC110 palmtop computer. 27.3% of this is non-standard words.

RFR
: To show that its not just geeky discussions that are full of non-standard words we extracted 209K tokens from the USENET group rec.food.recipes. 22% of these tokens are non-standard words.

The data was converted to a simple XML format and then each NSW was hand labelled (with a simple labelling tool, that included simple prediction of the type of the token).

The NSW types were

`EXPN`
: abbreviation, contractions e.g. adv, N.Y, mph, gov't

`LSEQ`
: letter sequence e.g. CIA, D.C, CDs

`ASWD`
: read as word, e.g. CAT, proper names

`MSPL`
: misspelling e.g. geogaphy

`NUM`
: number (cardinal) e.g. 12, 45, 1/2, 0.6

`NORD`
: number (ordinal) e.g. May 7, 3rd, Bill Gates III

`NTEL`
: telephone (or part of) e.g. 212 555-4523

`NDIG`
: number as digits e.g. Room 101,

NIDE
: identifier e.g. 747, 386, I5, PC110, 3A

NADDR
: number as street address e.g. 5000 Pennsylvania, 4523 Forbes

NZIP
: zip code or PO Box e.g. 91020

NTIME
: a (compound) time e.g. 3.20, 11:45

NDATE
: a (compound) date e.g. 2/2/99, 14/03/87 (or US) 03/14/87

NYER
: year(s) e.g. 1998 80s 1900s 2003

MONEY
: money (US or otherwise) e.g. \$3.45 HK\$300, Y20,000, \$200K

BMONY
: money tr/m/billions e.g. \$3.45 billion

PRCT
: percentage e.g. 75\%, 3.4\%

SLNT
: not spoken, word boundary e.g. word boundary or emphasis character: M.bath, KENT*REALTY, _really_, ***Added

PUNC
: not spoken, phrase boundary e.g. non-standard punctuation: "..." in e.g. DECIDE...Year, *** in $99,9K***Whites

FNSP
: funny spelling e.g. sllooooowww, sh*t

URL
: url, pathname or email e.g. http://apj.co.uk, /usr/local, phj@teleport.com

NONE
: token should be ignored e.g. ascii art, formating junk

Although at first glance these seem reasonable, after labelling we noted there is still some ambiguity on these. We did test our interlabeller agreement and found it very high but as a large percentage of this task is trivial the problems are always in that last few percent, and there we foudn out labellers we're consistent especially when it came to splitting things.

After labelling a section such as

> Today I bought a Sony NP-F530 1350maH. Like your 550 it is slightly larger than the native IBM battery pack. It's been 3 hours now on it's first charge - I am charging in the PC110.

will look like

> Today I bought a Sony<W NSW="LSEQ"> NP-F530,</W><W NSW="SPLT"><WS NSW="NUM"> 1350</WS><WS NSW="EXPN">maH.</WS></W> Like your<W

NSW="NIDE"> 550</W> it is slightly larger than the native<W NSW="LSEQ"> IBM</W> battery pack. It's been<W NSW="NUM"> 3</W> hours now on it's first charge - I am charging in the <W NSW="LSEQ"> PC110. </W>

Splitting of tokens only at white space, even for English was considered to be too limiting as when looking athe Festival token to word rules (or the equivalent int he Bell Labs systems) there seemed to be too many rules that were there just further split objects rather than expanding thing to words. Thus the splitter additional splits tokens that are letter/number conjctions, (forms of) mixed case conjunctions and things containing punctuation. The splitter is actually implemented as a set of regular expressions. But this too has a (in some sense) weird set of exceptions such as money symbols, urls, telephone numbers and some others.

The second stage was to assign the token tags to each split token. This is done (by default) from CART models trained from the the labelled data. For identifying the various alphabetic types (`LSEQ`, `ASWD` and `EXPN`) trigram letter language models were build that return an estimate of the probability that some alphabetic character sequences is a letter sequence, as pronouncable word or an abbreviation. The results are fed into the CART classifier, rather than used directly.

The full classifier accuracy various across domains ranging from 98.1% correct in the new data, to 91.8% in the PC110 (email) data.

Once classification is done a simple set of expanders is used to generate the words from the token plus type. This is true expcept for `EXPN` labelled tokens (abbreviations), as although they have been identified as abbreviations we still need to identify what they are an abbreviation of.

Finding out what the expansion of an abbreviation is can most simply be done by an abbreviation lexicon. But in some cases an new abbreviation may occur, that can be detected as an abbreviation, but isn't in the lexicon. This model attempts to solve this. It relies that given an abbreviation some full expansion of it appear somewhere in a corpus of the sort of text the text normalisation model will be applied to. There are two parts to this abbreviation expansion model. First a model that predicts all possible expnasions of an abbreviation. This is done with a WFST (weighted finite state transducer) model built from a trained CART model that predicts the deletion of characters from full words to form abbreviations. The second stage is a languge model that predicts the occurs of those full words in text. Together, for the classifieds domain, this works with about 20% error.

The advantages of the NSW model are

- Gives an explicit model
- Allows trainable models for domains
- Is equal to "standard" models (from Festival and the LDC text conditioning tools) for news data, and *much* better for all other domains
- Includes possibility of training from unlabelled data
- Designed as a TTS text analyser *and* as a simple text to word converter for text conditioning for langauge modelling for speech recongition or information retrieval etc

5.10 Two interesting text analysis problems

5.10.1 Tokenization without whitespace

In most European languages white space is used to separate basic words, but this isn't true in languages like Chinese and Japanese. There text is a continuous flow of characters. Punctuation is still often used but there is regularly no whitespace. Even newlines may be inserted within words to allow proper line up of characters. Within English this problme partly occurs in compound words and consequences of this can be seen in finding pronunciations of unknown words by letter to sound rules.

Consider the word `outhomer` (s baseball term). Festival's letter to sound rules, wrongly pronounce this as `aw th am m er`. Although we can see there is effectiver a word break between the letters `t` and `h` the letter to sound rules do not see this and map these two characters together as a single phone `th`. The reason we can seet `outhomer` as two words as the two words `out` and `homer`, is due to the high frequency of these individual words and the lower frequency of the full word pronounced as `aw th am m er`. As another example, consider the word `together`, it could be split as `to get her`, but isn't due the relative frequency of the full word over the not unusual trigram of `to get her`.

Thus the best *split* can be defined as most probable of all possible splits. Mathematically we can put this as the split that maximizes the probability of the sentences which we can estimate by

ProductOf foreach i in K P(wi | wi-1, ... wi-N+1)

Thus we use say tri (N=3) or bi (N-2) grams to estimate the probability of each word. This technique is used, in general for finding segmentations of Chinese and Japanese texts *sproat96b*. And it is quite successful.

However in its simplest form it needs a pre-tokenized database of words in order to collect the statistics for the ngrams. One technique to solve this is to use an iterating approach where an unsegmented database is segmented with some simple algorithm, say longest matching, then stats are collected and the statistically technique is used, and the stats are re-estimated. This process can be iterated until there no more improvements, on some held out test database.

5.10.2 Number pronunciation

In some languages, gender, case etc affect how numbers are proounced. That is the pronounced of the digit `1` depends on what it is refereing too. For example in Spanish

```
1 ni@~no --> un ni@~no (one boy)
1 ni@~na --> un ni@~na (one girl)
1 hermano --> un hermano (one brother)
1 hermana --> una hermana (one sister)
1 pais ---> un pais (one country)
1 ra'iz ---> una ra'iz (one root)
```

Although it might seem possible that checking if the following word ends in `a` or `o` might be a good

disambuguator there are many words in Spanish where gender cannot be easily identified from surface form and a lexicon is required. Further more the digit(s) may not be refering to the word immediately following that token and may ultimately require understanding of the sentence to get the right pronunciation.

Spanish is not the only language where declensions exist. In a Polish synthesizer we built some years ago, most of the work became concentrated on getting the pronunciation of numbers correct. This would be true for all slavic languages.

5.11 Exercises

1. Add a token to word rule, to say money values with two places after the point properly.
2. Add a token to word rule to say numbers in dates as ordinals (first, second, third, etc.) rather than cardinals. Also add a token to word rule to dates of the form "11/06/97" in their full form rather than number slash number ...
3. Use Sable markup to tell a joke.
4. Build a text mode for reading Latex, HTML, syslog messages, machine use summary or such like.

5.12 Hints

1. You will need to add a new definition for `token_to_words` by convention save the existing one and call that for things that don't match what you are looking for. Thus your file will look something like

```
(set! previous_token_to_words token_to_words)

(define (token_to_words token name)
   (cond
     ;; condition to recognize money tokens
     ;; return list of words
    (t
     (previous_token_to_words token name))))
```

   The actual condition and return list of words is similar to the treatment of email addresses described above. The regular expression for money is probably "$[0-9]+\\.[0-9][0-9]",
2. Take the above and add to it. The rule is probably: if the token is a two digit number and the succeeding token's name is a month name (or abbreviation of a month name) then return the word, first, second etc. You will need a new function to relate the digits to the pronunciation. If you don't know how to do that in Scheme the following will work

```
(define (num-to-ordinal num)
"Returns the ordinal in words for num (up to 40)."
(cdr
 (cdr
  (assoc (parse-number num)
```

```
'((1 first) (2 second) (3 third) ...
  (39 thirty-ninth) (40 fortieth))))))
```

Once you decide on the condition, remember that you need to return a *list* of words.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

6 Linguistic/Prosodic processing

This is the second stage in text to speech. We now have our words, what we need to produce is something to say. For this we will require segments (phonemes), durations for them and a tune (F0).

Although this can be in the most naive way, the quality of the speech, how natural, how understandable, how acceptable it is depends largely on the appropriateness of the phonetic and prosodic output. Prosody in language is used to add variation for emphasis, constrast and overall easy of unstanding and passing the message to the listener. The degrees of variation in prosody depend a lot on the language being spoken. Some things are lexical and cannot be changed without changing the words being spoken. For example lexical stress in English is part of the definition of a word, if you change the position of the stress you can change the word (e.g. from "pro'ject" (noun) to "proje'ct" (verb)). In other languages, such Chinese, tones play a part in lexical indentification thus changing (though not removing) the possible uses of F0 in utternaces.

Bad prosody makes utterances very difficult to near impossible to understand and so it is important for a speech synthesis system to generate appropriate prosody.

6.1 Lexicons

The simplest way to find the pronunciation of a word is to look it up in a word list or *lexicon*. This should give the syllabic structure, lexical stress and pronunciation in some phoneme set. For some languages (and for some words in some languages) this mapping may be trivial and there could exist a well-define mapping from the orthography to the pronunciation, Latin and Spanish are examples of languages where lexicons for pronunciation are mostly redundant. However in English this is not so trivial.

Some languages require analysis of the word into sub-parts or *morphs* removing their various modifications (usually prefixes and suffixes). For example in English we might only store the pronunciation of `move` in our lexicon and derive from it the pronunciation of `moved`, `moving`, `mover`, `movers`, etc. This a trade-off between lexicon size and amount of effort required to analyze a word. Languages such as German with significant compounding may require some form of morphological decomposition. Others with significant amount of agglutinative morphology (like Finnish and Turkish) cannot realistically be done without morphological analysis. Various semitic language like Arabic with vowel harmony require quite novel methods to analyze. See *ritchie92* for more details than you need.

The size and necessity of a lexicon is language dependent, but it is almost always convenient to have at least a small list for various acronyms and names etc. In fact although quite large lexicons (10,000s to 100,000s of words) are readily available for English, German and French, it is almost always the case that some words are missing from them and a localized lexicon of specific task specific words are required. This typically includes proper names, login names, etc.

Of course no matter how hard you try you are not going to be able to list every word. New ones are created often, certainly new names come into focus as world events happen. It is of course not acceptable for a speech synthesizer to fail to pronounce something so a fall back position that can guarantee pronunciation is important. Letter to sound rules can help with this. Often words can be pronounced reasonably from letter to sound rules. If the word isn't in the lexicon (i.e. this implies its not common), a guess at pronunciation might work, after all that's what humans need to do when they see a word for the first time. In some languages (e.g. Spanish) letter to sound rules can do the whole job for almost all words in the language.

In Festival a lexicon consists of three parts (all optional)

Compiled Lexicon
: A large list of words, their part of speech, syllabic structure, lexical stress and pronunciation. This is not loaded into the system but accessed as required and can be accessed efficiently even for lexicons with 100,000s entries

Addenda
: A typically smaller list of entries which are specific to a task or implementation. In Festival a typical addenda includes specialised words and names such as "ToBI", "PSOLA", "awb" and "email" which are not typically found in big lexicons.

Letter to sound rules
: A general letter to sound rule system is included in Festival allowing mapping of letters to phones by context-sensitive rules.

Multiple lexicons are supported at once as different lexicons may be required even for the same language, e.g. for dialectal differences. The phones produced by a lexicon should be suitable for the waveform synthesis method that is to be used (though Festival does supports phoneme mapping if really desired).

6.1.1 letter to sound rules

It is necessary for *all* words to be assigned a pronunciation but it is practically impossible to list all words in a lexicon. The solution to this is to offer a mechanism that assigns a pronunciation to words not found in the lexicon or agenda. For some languages where the orthography closes maps to pronunciation (e.g. Spanish) almost all pronunciation can be done by letter to sound rules.

The rule supported in Festival are fairly standard. The basic form of rule is

```
( LEFTCONTEXT [ ITEMS ] RIGHTCONTEXT = NEWITEMS )
```

For example

```
( # [ c h ] C = k )
( # [ c h ] = ch )
```

The # denotes beginning of word and the `C` is defined to denote all consonants. The above two rules which are applied in order, meaning that a word like `christmas` will be pronounced with a `k` which a word starting with `ch` but not followed by a consonant will be pronounced `ch` (e.g. `choice`.)

Rules are quite difficult to write but are quite powerful. Work has been done on learning them automatically both by neural net and statistical processes but those results have typically not been used in real synthesis systems. When building synthesizers for many languages and dialects it is necessary to be able to provide letter to sound rule sets quickly and efficiently so being about to automatically build rules from examples (e.g. a lexicon) is desirable.

black98b provides a basic trainable letter to sound rules system within the Festival framework. The idea is that from a reasonably sized lexicon a letter to sound system can be automatically generated. At repsent the process requires a little hand seeding but fairly minimal and unskilled. The results from applying this technique to English (UK and US), German and French are good and it seems very unlikely their quality godd be beaten by hand written rules without an awful lot of work (i.e. years of skilled labor).

The technique falls into two phases. First built alignments between letters and phones from the lexicon and secondaly build (CART) models to predict phones form letter contexts.

There are typically less phones that letters in the pronunciation of many langauges. Thus there is not a one to one mapping of letter t phones in a typical lexical pronunciation. In order for a machine learner technique to build reasonable prediction models we must first align the letter to phones, inserting epsilon where there is no mapping. For example

```
Letters: c  h  e  c  k  e  d
Phones:  ch _  eh _  k  _  t
```

Obviosuly there are a number of possibilities for alignment, some are in some way better than others. What is necessary though is to find these alignments fully automatically.

We provide two methods for this, one fully automatic and one requiring hand seeding. In the full automatic case first scatter eplisons in all possible ways to cause the letter and phoens to align. The we collect stats for the P(Letter|Phone) and select the best to generate a new set of stats. This iterated a number of times until it settles (typically 5 or 6 times). This is an example of the EM (expectation maximisation algorithm).

The alterantive method that may (or may not) give better results is to hand specify which letters can be rendered as which phones. This is fairly easy to do. For example, letter `c` goes to phones `k ch s sh`, letter `w` goes to `w v f`, etc. Typically all letter can at some time go to eplison, consonants go to some small number of phones and letter vowels got to some larer number of phone vowels. Once the table mapping is created, similary to the epsilon scatter above, we find all valid alignments and find the probabilities of letter given phone. Then we score all the alignments and take the best.

Typically (in both cases) the alignments are good but some set are very bad. This very bad alignment set, which can be detected automatically due to their low alignment score, are exactly the words whose pronunciations don't match their letters. For example

dept @tab d ih p aa r t m ah n t
lieutenant @tab l eh f t eh n ax n t
CMU @tab s iy eh m y uw

Other such examples are foreign words. As these words are in some sense non-standard these can validly be removed from the set of examples we use to build the phone prediction models.

A second compliation is that some letters more resonable rendered as more than one phone. For example

x as k-s @tab in `box`
l as ax-l @tab in `able`
e as y-uw @tab in `askew`

Currently these *dual phones* are simple added to the phone list and treated like any other phone, except they are split into the two individual phones after prediction.

CART trees are build for each letter in the alphabet (twenty six plus any accented characters in the language), using a context of three letters before an three letters after. Thus we collect features sets like

```
# # # c h e c --> ch
c h e c k e d --> _
```

Using this technique we get the following results.

OALD (UK English) @tab 95.80% @tab 74.56%
CMUDICT (US English) @tab 91.99% @tab 57.80%
BRULEX (French) @tab 99.00% @tab 93.03%
DE-CELEX (German) @tab 98.79% @tab 89.38%

The differences here reflect both the complexity of the language and the complexity of the particular lexicon itself. Thus the apparently much poorer result for US English (CMUDICT) over UK English (Oxford Advanced Learner's Dictionary) is due not only to OALD being done more carefully from a single source, but also the fact that CMUDICT contains many more proper names than OALD which are harder to get right. Thus although OALD appears easier on this held out data (every tenth entry from the lexicon), it is actually doing an easier task and hence fails more often of real unknown data, which more often contains proper names.

This brings into a number of notions of *measurement* that will be touched upon through this course. First how well does the test set reflect the data we actually wish to apply the model too. And secondly does the correctness measure actually mean anything. In the first case we note that different training/test sets even from the same dataset can give varying results. So much so that in LTS models its pretty difficult to compare any two LTS ruls sets except when they've been applied ot he same train/test set.

The second aspect of measurement is the question of how well does the notion of correct match what the system is actually going to do for real. In order to get a better idea of that we tested the models on actual unknown words from a corpus 39,923 words in the Wall Street Journal (from the Penn Treebank *marcus93*). Of this set 1,775 (4.6%) were not in OALD. Of those 1,360 were names, 351 were unknown words, 57 were American spelling (OALD is a UK English lexicon) and 7 were misspelling.

After testing various models we found that the best models for the held out test set from the lexicon we not the the best set for genuinely unknown words. BAsically the lexicon optimised models were over trained for that test set, so we relaxed the stop criteria for the CART trees and got a better result on the 1,775 unknown words. The best results give 70.65% word correct. In this test we judged correct to be what a *human listener* judges asa correct. Sometimes even though the prediction is wrong with respect to the lexical entry in a test set the result is actually acceptable as a pronunciation.

This also highlights how a test set may be good to begin with after some time and a number of passes and corrections to ones training algorithm any test set will be become tainted and you need a new test set. It is normal to have a *development test* which is used during development of an algorithm then keep out a real test set that only get used once the algorithm is developed. Of course as development happens in cycles the real test set will effectively become the development set and hence you'll need another new test set.

Another aspect to letter to sound rules that we've glossed over is the prediction of lexical stress. In English lexical stress is important in getting the pronunciation right and therefore *must* be part of the LTS process. Originally we had a separate pass after phone prediction that assigned lexical strees but following *bosch98* we tried to combine the lexical stress prediction with the phone prediction. Thus we effectively doubles the number of phone vowels by adding stressed and unstressed versions of them. This of course makes letter to phone prediction harder as there are more phones, but overall it actually gives better results as it more correctly predicts stress values.

6.2 Part of speech tagging

Note that a word's printed form does not always uniquely identify its pronunciation. For example, there are two pronunciations of the word *live*. Although by no means common, *homographs* (words written the same but pronounced differently) occur in most languages. The most common way to distinguish them is by part of speech--though that is still not enough for some pessimal cases (see discussion above about homograph disambiguation).

For example *wind* as a verb is pronounced `w ai n d` while as a noun it is `w i n d`. But this is not completely true as `w i n d` can be both a noun and a verb as can `w ai n d`. Rather than give up in despair we find that if we could label every word with its part of speech we can reduce the number of homograph errors significantly.

In Festival we have implemented a standard statistical part of speech tagger (*DeRose88*) which gives results around 97% correct tags for major classes. Comparing this against a test set of 113,000 words and a lexicon identifying homographs we find we make about 154 errors in pronunciation of homographs. There are significantly more spelling errors in these course notes than that, so the results are probably good enough.

But Festival still makes pronunciation errors. Pronunciation errors are very noticeable and memorable so they interfere with our understanding of the speech. Typical errors that still occur are

1. Letter to sound rules failing on novel forms.
2. Foreign proper names often fail but appear often in text.
3. Wrong part of speech identified, newspaper headlines are particularly difficult.
4. Part of speech is right but its not in the lexicon.
5. Part of speech not enough to differentiate pronunciation (and not yet dealt with by homograph disambiguation CART).

6.3 Prosodic phrasing

It is necessary to split utterance into prosodic phrases. When people speak they phrase their speech into appropriate phrase based on the content, how they wish to chunk things, and also how large their lungs are. Prosodic phrases probabaly have an upper bound, definitely less than 30 seconds and probably less than 15 seconds.

When looking break utterances into phrases in a text to speech system we have a number of options. Obviously breaks at punctuation are a reasonably place to start, but its not quite as simple as that.

bachenko90 describe a rule driven approach making use of punctuation, part of speech and importantly syntactic structure. This produces reasonably results but requires syntactic parsing to work, which is resource expensive and prone to error. One important finding is highlighted in this work. Phrasing tends to be balanced, that is we prefer to keep chunks roughly the same size, so factor like number words in phrase make a difference. Although prosodic phrasing is influenced by syntactic phrasing is can cross tranditionaly syntatic boundaries. For example a verb may bind with a subject or an object and will typical do so to the shorted or the two (to keep balance).

> (the boy saw) (the girl in the park) (the boy in the park) (saw the girl)

hirschberg94 uses CART trees and get excellent results (95% correct) for a simple corpus using features such as word, part of speech, distance from previous and to next punctuation.

ostendorf94 offer a stochastic method, still using CART trees but do many more experiments looking at the advantages and

disadvantages of different features. The also describe a hierarchical model to find the optimal prosodic labelling of different levels of breaks in utterances.

Festival supports two major methods, one by rule and the other using a statistical model based on part of speech and phrase break context *black97a*.

6.3.1 Phrasing by decision tree

The rule system uses a decision tree, which may be trained from data or hand written. A simple example included in `lib/phrase.scm', is

```
(set! simple_phrase_cart_tree
'
((R:Token.parent.punc in ("?" "." ":"))
  ((BB))
  ((R:Token.parent.punc in ("'" "\"" "," ";"))
   ((B))
   ((n.name is 0)  ;; end of utterance
    ((BB))
    ((NB))))))
```

This is applied to each word in an utterance. The models may specify any of three values (currently these names are effectively fixed due to their use in later models). NB stands for no break, B for small break, BB for large break. This model is often reasonable for simple TTS, but for long stretches with no punctuation it does not work well.

This model may take decision trees trained using CART so allows implementation of the technique discussed in *hirschberg94*. Experiments have been done like this but the results have never been as good as their's.

6.3.2 Phrasing by statistical models

Following the work of *sanders95* we have a more complex model which compares favourably with most other phrase break prediction methods.

Most methods try to find the likelihood of a break after each word but don't take into the account the other breaks that have already been predicted. This can often predict breaks at places where there is a much more reasonably place close by. For example consider the following two sentences

```
He wanted to go for a drive in.
He wanted to go for a drive in the country.
```

Although breaks between nouns and prepositions are probable a break should not occur between "drive" and "in" in the first example, but is reasonable in the second.

The more elaborate model supported by Festival finds the optimal breaks in an utterance, based on the probability of a break after each word (based on its part of speech context), and the probability of a break based on what the previous breaks are. This can be implemented using standard Viterbi decoding. This model requires

- Ngram model of B/NB distribution (up to a length of 8 or so seems optimal).
- Probabilities of breaks given current, previous and next part of speech tags.

See *black97a* for details of experiments using this technique.

6.3.3 Measuring phrasing algorithms

We again hit the problem of what is correct when it comes to phrasing. Almots nay phrasing can in some context be correct but it seems some phrasing for utterance sis more natural than others. When you comparing the output of a phrasing algorithm against some database where natural phrasing is marked some of the error will reflect just simply the freedom of choice in placing phrase boundaries while other parts of the error are unacceptable errors. Unfortuantely these different types of error can't easily be identified.

ostendorf94 uses a test set of multiple speakers saying the same data. They consider a prediction correct if there is any speaker who spoke that utterance with all the same phrase breaks. This attempts to capture some the random selection of possible phrasing for an utterance.

Note also how in different contexts different type of phrasing are appropriate. In conversational speech we are inclined to add many more break and use them more as a mechanism for information. In reading new text however we are much more likely to use a more standard phrasing. Thus when a synthesis system uses a phrase break algorithm trained from news speech it probabaly isn't appropriate for conversational speech. Thus the ability to change phrase break algorithm when the speaking style changes in important.

6.4 Intonation

There are probably more theories of intonation that there are people working in the field. Traditionally speech synthesizers have had no intonation models (just a monotone) or very poor ones. But today the models are becoming quite sophisticated such that much of the intonation tunes produced are often very plausible.

Intonation prediction can be split into two tasks

1. Prediction of accents: (and/or tones) this is done on a per syllable basis, identifying which syllables are to be accented as well as what type of accent is required (if appropriate for the theory).
2. Realization of F0 contour: given the accents/tones generate an F0 contour.

This must be split into two tasks as it is necessary for duration prediction to have information about accent placement, but F0 prediction cannot take place until actual durations are known. Vowel reduction is another example of something which should come between the two parts of tune realization.

The two sections may be done by rule or trained, Festival supports various methods to do this.

6.4.1 Accent assignment

hirschberg92 gives detailed (complex) rules to predict where accents go in utterances. *black95a* gives a comparison of Hirschberg's system and CART trees. The basic first approximation for accent prediction, in English, is to accent all stressed syllables in content words. This basically over assigns accents and is most noticeably wrong in compound nouns. But even this simple heuristic can be around 80% correct.

Accent type is a slightly different problem. The ToBI (*silverman92*) intonation theory states that there are only a small number of distinct accents in English, about 6. Therefore, we need to then choose between these 6 accents types. Although

there seems to be some linguistic difference in the use of each accent its difficult to fully identify it. Statistically trained methods (again CART) produce reasonable but not stunning results.

Some theories, like ToBI make distinctions between accents and boundary tones. Tones are the intonation events that occur at the end (or start) of prosodic phrases. The classic examples are final rises (sometimes used in questions) and falls (often used in declaratives).

Festival allows accent placement by decision tree. A rather complex example trained from American news speech is in `FESTIVALDIR/lib/tobi.scm' which shows the sort of tree you need to get reasonable results.

A much more simple example is just to have accents on stressed syllables in content words (and single-syllable content words with no stress). A decision tree to do this is as follows

```
(set! simple_accent_cart_tree
 '
  ((R:SylStructure.parent.gpos is content)
   ((stress is 1)
    ((Accented))
    ((position_type is single)
     ((Accented))
     ((NONE))))
   ((NONE))))
```

6.4.2 F0 generation

Again there are various theories for F0 generation, each with various advantages and disadvantages. This is of course language dependent but a number of methods have been relatively successfully over multiple languages.

fujisaki83 builds an F0 contour from parameters per prosodic phrase and has good results for simple declarative (Japanese) sentences. *taylor94b* has a more general system which can automatically label contours with parameters and generate contours from these parameters for a wide range of intonation tunes (work is progressing to integrate this into Festival *dusterhoff97a*). The ToBI system says little about generation of the F0 contour from their higher level labels but *black96* gives a comparison of two techniques. ToBI as yet has no auto-labeller (though there are people working on it).

Note that an auto-labelling method is desirable because it can generate data from which we can train new models.

This task of generating a contour from accents is actually much better defined than generating the accent placement or their types.

Festival supports a number of methods which allow generation of *target* F0 points. These target points are later interpolated to form an F0 contour. Many theories make strong claims about the form of interpolation.

The simplest model adds a fixed declining line. This is useful when intonation is be ignored in order test other parts of the synthesis process.

The General Intonation method allows a Lisp function to be written which specify a list of target points for each syllable. This is powerful enough to implement many simple and quite powerful theories.

The following function returns three target points for accented syllables given a simple hat pattern for accents syllables.

```
(define (targ_func1 utt syl)
  "(targ_func1 UTT ITEM)
Returns a list of targets for the given syllable."
  (let ((start (item.feat syl "syllable_start"))
        (end (item.feat syl "syllable_end")))
    (if (not (eq? 0 (length (item.relation.daughters syl "Intonation"))))
        (list
         (list start 110)
         (list (/ (+ start end) 2.0) 140)
         (list end 100)))))
```

Of course this is too simple. Declination, change in relative heights, speaker parameterization are all really required. Besides what height and width should accents be?

Based on work in *jilka96* there is an implementation of a ToBI-by-rule system using the above general intonation method in `lib/tobi_f0.scm'.

A different method for implementing ToBI is also implemented in Festival and described in *black96*. In this model, linear regression is used to predict three target values for every syllable using features based on accent type, position in phrase, distance from neighbouring accents etc. The results appear better (both by listening and measuring) than the rule approach. But of course in this case we need data to train from which we do not always have.

6.4.3 Some intonation models

This section gives a basic description of a number of different computational intonation theories with their relative advantages and disadvantages.

ToBI (Tones and Break Indices) *silverman92* is a intonational labeling standard for speech databases that in some way is based on Janet Pierrehumbert's thesis *pierrehumbert80*.

The labelling scheme consists of:

- 6 discrete intonation accents types: `H*`, `!H`, `L*`, `L*+H` and `L+H*`.
- 2 phrase accent type: `H-` and `L-`
- 4 boundary tones: `L-L%`, `L-H%`, `H-L%` and `H-H%`
- 4 break levels: 1, 2, 3, and 4
- A `HiF0` marker for each intonational phrase

A ToBI labeling for an utterance consists of a number of tiers. The accent and boundary tone tier, the break level tier and a comment level tier.

As is basically stands ToBI is not designed as computational theory that generation allows of an F0 contour from a labelling, or generation of labels from an F0, but it has been used as such. *anderson84* offer a rule based method for generating F0 contours from a basic ToBI-like labelling. Their basic structure is to define a set of basic heights and widths for each accent type. A baseline which declines over time is defined and the accents are placed with respect to this baseline. All values in their initial implementation were hand specified.

Another implementation, again basically by rule, is *jilka96*. An implemenation of this rule system is given in the festival distribition in `festival/lib/tobi_rules.scm'.

The standard voices in festival also use a basic ToBI framework. Accents and boundary types are predicted by a CART tree, but the F0 generation method is a statistically trained method as described (and contrasted with *anderson84*) in *black96*. In this case three F0 values are predicted for each syllable, at the start, mid vowel and end. They are predicted using linear regression based on a number of features including ToBI accent type, phrase position, syllable position with contexts. Although a three point prediction system cannot capture all the variability in natural intonation, by experiment it has been used to be sufficient to produce reasonable F0 contours.

The Tilt Intonation Theory *taylor00a*, in some sense takes a bottom up approach. Its intention is to build a parameterization of the F0 contour, that is abstract enough to be predictable in a text to speech system. A Tilt labelling of an utterance consists of a set of accent and boundary labels (identified with syllables). Each accents has 4 continous parameters which characterize the accents shape position etc. These parameters are:

- Duration
- Amplitude
- Peak position: distance from start of accent to beginning of the vowel in the related syllable
- Tilt parameter: a continuous value from -1.0 to 1.0, where -1.0 denotes a an accent which solely rises with now fall, 0.0 an accent with equal rise and fall, and 1.0 represents an accent which no rise and all fall. Numbers with the range vary the rise/fall ratio accordingly

Also each intonation phrase has a startF0 value.

It has been shown that a good representation of a natural F0 contour can be made automatically from the raw signal (though it is better of the accents and boundaries are hand labelled). *dusterhoff97a* further shows how that parameterization can be predicted from text.

The Fujisaki model *fujisaki83*, is again a bottom up data-driven model. It was originally developed for Japanese, though also has a German model *moebius96*. In the Fujisaki Model intonational phrases are represented by an phrase command and an accent command, and number of accent commadns may exist in each phrase. Hear a command" consists of an amplitude and a position. The model is a critically damped impluse modul where the F0 decays over time based on the enerygy inserted by phrase and accent commands. This model is argued to be physiologically based.

Moehler *moehler98* offers a model that uses both the nothings of data-driven as in Tilt and the Fujisaki model, but also the advantages of a phonology model like ToBI. He paramterizes the F0 in a way not-dissimilar to Tilt, but importantly then clusters differnt types of intonation contours by vector quanitization. Thus he has say 8 different types of accent that are data derived. Experiments in *syrdal98a* show this method to be better than Tilt and the *black96* model.

Ross *ross96* uses a dynamical model to represent the F0 and has good results on the Boston University FM Radio corpus.

A final method worth mentioning which offer much promise is to use the selection natural contours from aprorpriate speech examples rather than predict a new one. Using techniques very similar to what is becoming common in waveform (segmental) unit selection *malfrere98*.

Although tests have often been made to show how a theory may be better than some other thoery, actually the results of such tests should be not taken as absolute. In all cases the people doing the tests spend more time on implementing and tuning a particular theory than its competitors. I'm not writing this because *syrdal98a* shows other work to be better than work done previously by me, I'm as guilty of this in *black96* where I implemented both techniques. However although broad ordering of theories is probabaly correct depending too much on the F0 RSM error and corelation is almost certainly too naive a measure for true qualitative measures of a theory's correctness.

6.4.4 F0 Extraction

We have simply stated that intonation invovles modelling the F0 (underlying tune) of an utetrance. In this section we'll be more specific about what we means by F0 and how one can find it in speech.

The F0 is the basic tune in speech for mails it usually ranges between about 90Hz and 120Hz and about 140Hz to 280Hz in females. In general an F0 starts higher at the begining on a sentence and gets lower of time, reflect the depletion in the air rate as we speak, though it is possible to make it rise over time. In English, accents will appear at important words in the sentence, though their shape, amplitutde, duration and exact position will vary.

A typical F0 will look like

The F0 is in fact not as smooth as this. First, there are only pitch periods during voiced speech, second certain segments (particularly obstruents such as stops), can locally affect the pitch so that the underlying natrual paitch may actaully look like the following rather than the above.

Note that there is no F0 during the /f k/ as these are unvoiced. Also not the somehwta variable F0 valuse around the /hh/ of "heads" which although we may state is unvoiced actually as pitch periods near it.

When modelling F0 it is useful to have a clea prepresentation of it. As extraction methods of F0 from raw wave signals are prone to error we also, where exact F0 representation is important, we can also record the electrical activity in the glottis and predict F0 from that. We do this with an electro-glottograph (EGG) some called a larygnograph though first name is easier to say. The device consists of two small electrodes that are strapped to the throat while recording. The signal, when listened to sounds like a muffled buzz (cf. Kenny from South Park). The following shows a waveform signal with the corresponding EGG signal for the nonsense word /t aa t aa t aa/.

If we zoom in we can see the EGG signal (on the bottom) is much cleaner that the top one. Though it should be noted this is a particularly clean signal.

Extracting F0 from waveforms is not as difficult as speech recognition but is still a non-trival problem (*hess83*). Most algorithm require (or at least work much better with) knowledge of the range of the speaker. Though frequency doubling and halving is still a common type of error. One of the standard tools in pitch detection algorithms is *autocorelation*.

There are two basic uses of F0 extraction, first to get the whole contour. In this case smoothing is almost certainly required to filling unvoiced sections (by interpolation) and to smooth out segmental perturbations (so-called micro-prosody). In this case what is required is a track of values at fixed intervals (say 10ms) giving the approximate F0 values throughout the utterance.

The second use of F0 extraction is in signal process which is dicussed in detail later. In this case each *pitch period* is indetified exactly. Many signal processing techniques that can modify pitch and duration independantly work on the pitch periods of a signal. It is easier to get an estimate of the pitch over a section of waveform than to reliably get the pitch period properly identified hence F0 modelling can (and is) often done from pitch contours extracted from a waveform but it is

harder to do signal processing cleanly without an EGG signal, evn though it is desirable not to rely on the extra piece of informatiom.

6.5 Duration

In all of the current synthesis techniques we will talk about, explicit segmental durations are necessary for synthesis. Again many techniques have been tried, many of which are supported by Festival.

The easiest method is to use a fixed size for all phones. For example 100 milliseconds. The next simplest model assigns the average duration for that phone (from some training data). This actually produces reasonable results. However you can quickly hear that phones in an average only model sound clipped the start and end of phrases. Phrase final lengthening is probably the most important next step.

The Klatt duration rules are a standard in speech synthesis. Based on a large number of experiments modifications from a base duration of all the phones are describe by a set of 11 rules, *allen87*. These modify the basic duration by a factor based on information such as position in position in clause, syllable position in word, syllable type, etc. The results sound reasonable though a little synthetic. These rules are one of the fundamental features that people subconsciously identify when recognizing synthetic speech.

Others have tried to look at predicting durations based on syllable timings rather than the segmental effect directly *campbell91*. They've had some success and is an area actively studied in duration prediction.

In Festival we support fixed, average and duration modified by rule. The Klatt rules are explicitly implemented. Our best duration modules are those trained from database of natural speech using the sorts of features used in the Klatt rules but deriving the factors by statistical methods.

A simple duration decision tree predicts a factor to modify the average duration of a segment. This tree causes clause initial and final lengthening as well as taking into account stress.

```
(set! spanish_dur_tree
 '
   ((R:SylStructure.parent.R:Syllable.p.syl_break > 1 ) ;; clause initial
    ((R:SylStructure.parent.stress is 1)
     ((1.5))
     ((1.2)))
    ((R:SylStructure.parent.syl_break > 1)   ;; clause final
     ((R:SylStructure.parent.stress is 1)
      ((2.0))
      ((1.5)))
     ((R:SylStructure.parent.stress is 1)
      ((1.2))
      ((1.0))))))
```

6.6 Post-lexical rules

There are some segmental effects which cannot be determined for words in isolation. They only appear when the pronunciations are concatenated. May of these co-articulatory effects are ignored in synthesis which often leads to what appears to be over-precise articulation.

One effect that can however be easily dealt with is vowel reduction. The effect often happens in non-important function words. Although this can be trained from data we can specify mappings of full vowels to schwas is specific contexts.

Another form of reduction is word contractions, as is "it is" to "it's". This is very common in speech and makes a synthesizer sound more natural.

Festival supports a general method for post-lexical rules. These may be specified on a per-voice basis. Although some phenomena occur across dialects some are dialect specific.

For example in British English, post-vocalic "r" is deleted unless it is followed by a vowel. This can't be determined in isolated words for word final r's so this is done as a post-lexical rule. We apply the following simple CART to each segment in the segment relation after lexical look up to decide if the "r" has to be deleted or not.

```
(defvar postlex_mrpa_r_cart_tree '((name is r)
   ((n.ph_vc is -)
    ((delete))
    ((nil)))
   ((nil)))
```

6.7 Summary of Linguistic/Prosodic processing

Linguistic/Prosodic processing is the translation of words to segments with durations and an F0 contour.

Specifically we have identified the following modules

- Lexicons: mapping words to pronunciations
- Letter to sound rules: when no list of words is available
- Intonation: finding the tune
    - Accent assignment: where are the accents and what are their type.
    - F0 contour generation: by rule or statistical method.
- Duration: specification of length of each segment.
- Post-lexical rules: co-articulatory effects between words.

Although all of these can implemented by simpel rules, they can be trained (to some degree) from data. In is generally agreed that data driven models are better though often in specificase hand written rules will win. Note it is much easier to tweak a rule to cater for a particaulr phenomena that tweak a statistically model to better highlight a rare phenomena in a traing set.

A important aspect to measuring correctness of linguistic and prosodic models is to realise the distinction between existing mathematical measures and human perception. For example in measuring a duration model we can find the root mean squared error and correlation of a prediction model with respect to a test set. Those figures can then be used to judge how good a duration model is. But we are assuming that these are corelated with human perception which although maybe broadly true is almost certainly an over-simplication. Mathematical models which properly corelate with human perception are necessary for proper prosodic models but at present are not easily available. It is often said that synthesis should use more human listening tests in evaluation but note that developing preoper human evalation tests is in itself difficult and values received from them are also prone to error as they still are indirect measures of human perception.

6.8 Exercises

1. Build an intonation system that adds small hat accents to all stressed syllables, without declination, except on final syllables, which rise for a question and fall for all other sentence types.
2. Write a Festival script that read arbitrary text from standard input and output it on standard output giving newlines only at the end of prosodic phrases.

6.9 Hints

1. You need to use the General Intonation system to build this. You'll need an accent assignment decision tree and a Scheme function predict the targets for each syllable. The accent assignment tree is as mentioned above

```
(set! int_accent_cart_tree
 '
  ((R:SylStructure.parent.gpos is content)
   ((stress is 1)
    ((Accented))
    ((position_type is single)
     ((Accented))
     ((NONE))))
   ((NONE))))
```

   And you'll need to write a function that generates the F0 targets, something like

```
(define (targ_func1 utt syl)
  "(targ_func1 UTT ITEM)
Returns a list of targets for the given syllable."
  (let ((start (item.feat syl "syllable_start"))
        (end (item.feat syl "syllable_end")))
    (cond
     ((string-matches (item.feat syl "R:Intonation.daughter1.name") "Accented")
       (list
        (list start 110)
        (list (/ (+ start end) 2.0) 140)
        (list end 100)))
      (
        ;; End of utterance as question
        ;; target for mid point and high end pont
      )
      ( ;; End of utterance but not question
        ;; target for mid point and low end pont
      ))))
```

   The condition `(equal? nil (item.next syl))` will be true for the last syllable in the utterance and `(string-matches (item.feat syl "R:SylStructure.parent.R:Word.last.R:Token.parent.punc") "\\?")` will be true if there is a question mark at the end of the utterance. You also need to set up these functions as the intonation method

```
(Parameter.set 'Int_Method 'General)

(set! int_general_params
      (list
```

```
        (list 'targ_func targ_func1)))
```

2. Again you need to copy the `text2pos' script discussed in the previous set of exercises. This time you must add a call to the `Phrasify` module to the function `find-pos`. In the loop that extract information about each word in an utterance you must add a new feature `pbreak` to give the value of break after that word. The function itself (with the `printfp`) print out each word and a space (the variable `space` contains a space that prints without quotes round it) but only call `terpri` when the `pbreak` value is `B` (or twice when the value is `BB`).

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

7 Waveform synthesis

The third major part of the synthesis process is the generation of the waveform itself from some fully specified form. In general this will be a string of segments (phonemes) with durations, and an F0 contour.

There are three major types of synthesis techniques

articulatory
: model human vocal tract, high quality, but computationally expensive and difficult to find good parameters. This is currently not so good for consonants.

formant
: as found in DEC talk, this tries to model the signal itself. This produces reasonable quality but sounds synthetic. Its also difficult to find the good parameters for it.

concatenative
: use pieces of natural speech and puts them together. Diphones and nphones are most popular but longer units and larger inventories are becoming more popular. The selected units also need to be modified, prosodically (and spectrally), to match the targets by some signal processing technique

We will only talk about concatenative synthesis here. many of the more practical aspects of waveform synthesis are discussed in the Festival Document (http://festvox.org). That document includes detailed instructions and code for building new diphone synthesizers

7.1 Diphones

Concatenating phonemes has the problem that all the joins happen at what is probably the least stable part of the waveform: the part changing from one phone to the next. Concatenating diphones (middle of one phone to middle of the next) means joins (may) happen in a more stable part of the signal. This effectively squares the number of units required but this can still be feasible. This number phones required for different languages various significantly. For example Japanese has around 25 phones giving rise to about 625 diphones while English has around 40 phonemes giving rise to about 1600 diphones.

Of course not all diphones are really the same and it has been shown that selecting consonant clusters can be better than strictly diphones. Most "diphone" synthesizers today include some commonly occurring triphone and nphone clusters.

As an inventory containing single occurrences of diphones will not provide the desired prosodic variation,

the diphones are modified by some form of signal processing to get the desired target prosody.

7.1.1 Diphone schema

There are two methods in collecting diphones. The diphones may be extracted from a set of *nonsense words* or from a set of *natural words*.

Generating nonsense words that provided diphone coverage in a language is relatively easy. A simple set of carrier words is constructed and each phone-phone combination is inserted. For example qw might use the carrier

```
pau t aa C V t aa pau
```

where each consonaant and vowel is inserted. Typically we need slightly different carriers for the different combinations, consonant-vowel, vowel-consonant, vowel-vowel, consonant-consonant. We also need carriers for silence-phone and phone-silence too.

In most languages there are additional phones (or allophones) which should be included in the list of diphones. In some dialects of English /l/ has significantly different properties when in onset positions compared with coda position (light vs dark L). In US English flaps are common as a reduced form on /t/. These require there own carrier words and a concrete definition of where they do and do not occur (which may vary between particular speakers).

Much less specific consonant clusters should also be considered and given their own diphone. In our English voices we typical include both an `s-t` diphone for the syllable boundary case (as in `class tie` as well as a `s_-_t` diphone for use of `st` in a consonant cluster (e.g. `mast`). Some other diphone systems introduced "diphones" for longer units thus instead of have a specail type of `s` for use in consonant clusters they introduce a new pseudo-phone `st`.

The addition of extra diphones over and above the basic standard is a judgement based on the effect it will have synthesis quality as well as the amount of extra diphone it will introduce. As the speaker will tire over time it is not possible to simple add new allophones and still have a manageable diphone set (e.g. adding stress vowels will almost double the number of diphones in your list).

As we stated there are two ways to collect diphones, first as described above from nonsense words and the second is from a list of natural words. Most of the points above apply to either technique. As yet there is no conclusive prrof that either technique is better than the other as it would require careful recording of both types in the same environment. As recording, labelling and tuning is expensive from a resource point of view few people are willing to test both methods but it is interesting to point out the basic advantages and disadvantages of each.

The nonsense word technique as the advantage of being simpler to ensure coverage. Words are specified

as phone strings so you know what exactly how the words should be pronounced. The main disadvantage of the nonsense word technique is that ther eis a loss of naturalness in their delivery. The nonsense words are unfamiliar to the speaker and there for are produced as natural and real words would be. This is probably one factor that makes diphone synthesizers appear to lack a certain amount of naturalness. Though the main factor is probably that the speaker must deliver some 1500 words is such a controlled environment that its is not surprising that the resulting voices sound bored.

The natural word technique has the possibility of provide more natural transition. There are problmes though, its is difficult to prompt a user to pronounce those words in the intended form. If flaps are required (or not required) it can be harder to coach the speaker to say the intend phone string. Also because the list is of natural words the speaker will say them in a more relaxed style which may make the overall quality less consistent and therefore less stiable for concatenation and prosodic modification. It is also harder to find natural words that are both reasonably familar, have standard pronunciations and properly cover the space, especially when working in new languages. But the relative merits of these techniques have not really been fully investigated.

In more recent collections of diphone databases, following OGI's method, we currently recommend that diphones (at least as nonsense words) are delivered at as near constant articulation as possible. Thus they are recorded as a monotone with near fixed durations and near constant power. There reasoning behind this is that as there segments are to be joined with other segments we wish them to be as similar as they can be to minimise distortion at join points.

7.1.2 Recording diphones

There are ideals for a recording environment, an anechoic chamber, studio quality recording equipment, and EGG signal and a professional speaker. But in reality this ideal is rarely reached. In general we usually settle for a quiet room (no computers, or air conditioning), and a pc or laptop. A headmounted microphone is very important and so cheap to provide it is not worth giving up that requirement. The problem without an headmounted mike, the speaker's position may move from the (handheld or desk mounted) microphone and that change introduces acoustic difference that would be better to avoid.

Laptops are relatively good for recording as they can be taken to quiet places. Also as power supplies and CRT monitors are particularly (electrically) noisey a laptop ran from batteries is likely ot be a quieter machine for recording. however you also need to take into account that the audio quality on laptops is more variable than on desktops (also it is much easier and practical to change the sound board on a desktop than on a laptop. Although machines do always generate electrical noise, the advantage of using a computer to directly record utterances into the appropriate files is not to be under estimated. Previously we have recorded to DAT tape and later downladed that tape to a machine. The processes of downloading and segmenting into the appropriate files is quite substantial (at least a number of days work). Give how much of the rest of the process is automatic that transfer was by far the most substantial part and the benefits of a better recording may not in fact be worth it.

However the choice of off-line recording should be weighed properly. If you are going to invest six months in a high-quality voice its probably worth the few extra days work, while if you just need a reasonable quality voice to build a synthesizer for next week recording directly to machine *after ensuring you are getting optimal quality* is probably worth it. But in using a macine to record it is still paramount to ensure a headmounded microphones *and* proper audio settings to minimise channel noise.

7.1.3 Labelling diphones

Once diphone recording have been made the next task is to label them. In the case of nonsense words labelling is much easier than phonetic labelling of continuous speech. The nonsense words are defined as string of particular phones, and hence no decision about what has been said it necessary only where the phone start and end is required. However it is possible that in spite careful planning and listening the speaker has pronounced the nonsense word wrongly and that should be checked for but this is relatively rare.

In earlier cases we would hand labelled the words. As stated this isn't too difficult and mostly just a laborious task. However following *malfrere97*, we have taken to autolabelling the nonsense words. Specifically we use synthesized prompts and time align generated cepstrum vectors with the spoken form. The results from this are very good. Most human labellers make mistakes and hence even with human labeller multi-passes with corrections are required. The autolabeller method actually produces something better than an initial human labeller and takes around 30 minutes as opposed to 20-30 hours.

Even when you are collecting a new language for which prompts in that language cannot be generated the autolabelling technique is still better than hand labelling. In the nonsense word case the phone string is very well defined and it can be alighnment to even an approximation of the native pronunciation. For example we took a Korean diphone databases and generated prompts in English. Even though Korean has a number of phones for which ther eis no equivalent in English (asperated verses unasperated stops) the alignment works sufficiently well to be practical. Remember the alignment doesn't need to detect the difference between phones only how best can an existing phones list be aligned to another.

However although the autolabelling processing is good it still benefits for hand correction. In general the autolabelling will mostly get the whole word correct but occasionally (less than 5% of the time) the whole word is wrongly labelled. Often the reason can be identified such as a lipsmack causes a /t/ to be anchored in the wrong place. But other times there is no obivous reason for the error. In these cases hand corection of the labels is required.

It would be good if we could automatically identify where the bad labels are so they can be presented for correction. This isn't as self-referential as it first appears. Although we've not released the tools to do this we've made initial investigations of techniques that can detect when labelled phones are unsual. The simplest technique is to find all phones whose duration is more than three standard deviations from the mean. This often points to erroneously labelled utterance. Secondly we note that findi difference in the power and frequency domain poarameters of phones outside a 3 standard deviation range also indetify

potential problems.

Irrespective of the labelling trechnique what we require is finding phone bounraries with an acuracy of about 10ms, though actually up 20ms may be sufficient. More importantly though we need to know when the stable part of phones are so we can place the diphone boundary. In the autolabelling case we can include the known diphone boundary (stable point) that exists in the prompt in the label file, thus autolabelling gives not only the phone boundaries but also the stable part as identified in the prompt.

When no boundaries are explicitly available (e.g. when cross language alignment is done or labels are pre-existing), we use the following rules:

1. For stops: stable point is one third in. This should be within the slince part of the stop.
2. For phone-silence: stable point is one quarter in. Phones before silence have much less power than phones within words. Thus is a half point was selected as the stable point a larger power difference would occur when this diphone is joined to another.
3. For all other diphone: stable point is 50% in.

Note these are guessed, they are not optimal. In you invetsigate the best diphone synthesizers out there you will find that the stable points have all been hand labelled, very carefully by looking at spectragrams and linstening to joins. Such careful work does improve the quality. In fact most of the auto-labelled voices we have done included a fair amount of careful labelling of stable points (though we now benefit from that when re-labelling using these dbs as prompts).

Rather than using hueristics more specific measurements for finding the optimal join point can be done. *conkie96* descirbes a method for find optimal join poitns by measures the cepstral distance as potential join points. Either using the best (smallest distance) for each possible join or an average best place can be selected. Such a technique is used at run time in the unit selection techniques discussed below.

7.2 Getting longer Units

Although selecting diphones will give a fair range of phonetic variation it will not give the full prosodic or even pectral range that is really desired. If we could select longer units, or more appropriate units for a database of speech the resulting concatenated forms will not only be better they will require less signal processing to correct them to the targets (Signal processing introduces distortion so minimising its use is usually a good idea). We could use finer detailed phonetic classification: such as stress v. unstressed vowels thus increasing the size of our inventory explicitly.

Sagisaka, (*nuutalk92*) used a technique to select *non-uniform units* from a database of isolated words. These are called *non-uniform* because the selected units may may be phone, diphone, triphones or even longer. Thus the inventory contained many examples of each phone and acoustic measures were used to find the most appropriate units. In many cases multi-phone units would be selected especially for common phonetic clusters.

This work was only for Japanese, but had specifically excluded any prosodic measures in selection (namely duration and pitch), even though most people agreed that such features would be relevant. Further work in Japan extended the ideas of *nuutalk92* and *campbell92b*, to develop a general unit selection algorithm that found the most appropriate units based on any (prosodic, phonetic or acoustic) features *black95d*, implemented in the CHATR system.

hunt96 further formalised this algorithm relating it to standard speech recognition algorithms and adding a computationally efficient method for training weightings for features. Other techniques for selecting appropriate units are appearing.

donovan95 offers a method for clustering examples units to find the most prototypical one. *black97b* also uses a clustering algorithm but like *hunt96* then finds the best path through a number of candidates. Microsoft's Whistler synthesizer also uses a cluster algorithm to find the best example of units (*huang97*).

In the following sections we will particularly describe the work of *hunt96*, *black97b* and *taylor99b* as three typical selection algorithms. They also are the ones we are most familiar with. *hunt96* was implemented in the CHATR speech synthesis system from ATR, Japan. As the *hunt96* based selection algorithm is proprietary no implemention is available in festival but an implementation of *black97b* is freely available.

7.2.1 Finding the best unit

What we need to do is find which unit in the database is the *best* to synthesize some target segment. But this introduces the question of what "best" means. It is obviously something to do with closest phonetic, acoustic and prosodic context but what we need is an actual measure that given a candidate unit we can automatically score how good it is.

7.2.2 Acoustic cost

The right way to do this is to play it to a large number of humans and get them to score how good it is. This unfortunately is impractical, as there may be many candidates in a database and playing each candidate for each target (and all combinations of them) would take too long (and the humans would give up and go home). Instead we must have a signal processing measure which reflects what humans perceive as good synthesis.

Specifically we want a function that measures how well a given a unit, can be inserted into a particular waveform context. What we can do is set up a psycho-linguistic experiment to evaluate some examples so that we have a human measure of appropriateness. Then we can test various signal processing based distance measures to see how they correlate with the human view.

hunt96 propose a cepstral based distance, plus F0 and a factor for duration difference as an acoustic measure. This was partly chosen through experiments, but also by experience and refinement. It is this *acoustic cost* that will be used in define "best" when comparing the suitability of a candidate unit with a target unit. *black97b* use a similar distance but include delta cepstrum too. *donovan95* use HMMs to define acoustic distances between candidate units.

7.2.3 Hunt and Black Target Cost

During synthesis, of course, we do not have the waveform of the example we wish to synthesize (if we did we wouldn't need to synthesize it). What we do have is the information in the target that we must match with the information in the unit in the database. Specifically the target features will only contain information that we can predict during the synthesis process. This (probably) restricts it to phonetic and prosodic features to

- phonetic context
- duration
- pitch
- position in syllable, word, and phrase.

Thus excluding spectral properties.

We do have the spectral properties of the candidate units in the database, but can't use them as the targets don't have spectral information to match to. We can however derive the phonetic and prosodic properties of the candidate units so we can use these to find a cost.

Specifically, both target units and candidate units are labelled with the same set of features. For each feature we can defined a distance measure (e.g. absolute, equal/non-equal, square difference). And we can define the *target cost* as a weighted sum of difference of features.

Of course this is not the same as the acoustic cost nor do we know what the weights should be for each feature. But we can provide a training method which solves that problem.

Experiments with presenting pitch, duration etc in different domains (e.g. log domain) are worthwhile as we do yet know the best way to represent this information.

7.2.3.1 Estimating the acoustic cost

As stated above we define the target cost to be the weighted sum of differences of the defined features, and we wish that the target cost is correlated with the acoustic cost.

If we consider the special case where we compare known examples from the database we are synthesizing

from: in this case we have the target features and the candidate features *but also* we can calculate the acoustic cost.

As we have in this "resynthsis case" both the acoustic cost and the target and candidate features the only thing missing is the weights for the feature distance. We can thus use standard techniques to estimate the weights.

7.2.3.2 Hunt and Black weight training

Weights are found for groups of phones, e.g. high vowels, nasals, stops etc.

For each occurrence of a unit in the database find its acoustic distance with all other occurrences (of the same type). Also find its target cost distances for each feature. This will produced a large table of acoustic cost and list of features distances of the form

```
AC = W0 d0 + W1 d1 + W2 d2 ...
```

That only the Wn are unknown. We can then estimate Wn for the class of units using linear regression.

7.2.4 Black and Taylor Unit Clustering

While *hunt96* finds appropriate candidates by measuring the *target cost* at synthesis time, *black97b* effectively moves that costing into the training processing.

Similar to *donovan95*, *black97b* clusters the occurrences of units into clusters of around 10 to 15 examples. Clustering is done to minimise the acoustic distance between members using the acoustic distance defined in the the previous section. The clusters are found by partitioning the data based on questions about features that described units. Theses features are only those that are available at synthesize time and that are available on target units. Such features include phonetic context, prosodic context, position in syllable, position of syllable in word and phrase etc. The same sorts of features that are used in target costs for *hunt96*. The target cost in this case is the distance of particular unit to its cluster centre.

In this case the units are sorted into clusters during training rather than measured at synthesis time offering a more efficient selection of candidate units.

In later version of this work actual diphones are selected rather than full phones. This leads to better classification, and better synthesis.

7.2.5 Taylor and Black: Phonological Structure Matching

In both the above techniques selection is based on a per phone (or diphone) basic. This may not be optimal as if there are longer segments that are apropriate they will only be found if the appropriate target costs (or cluster distances) are set up. In PSM a more explicit method is used to find appropriate longer units. In this case the database used for sleection is labelled with a tree based representation.

Synthesis requires matching a tree presentation of the desired utterance against the databases. This is done top down looking for the best set of subtrees that can be combinded into the full new utterance. The matches move furtehr and further down the trees looking for a reasonable number of alternatives (say 5) if such a number is not founc matches are attempted and the next level down, until phone level matches are made.

Specifically this technique has the advantage of finding longer matches with less combutation (and dependence of target costs of adjacent phones in the database being similarly scaled. Also as this tree method, although it may include various features, can primarly be done with phones (and context, stress levels, accents syllable etc. This is a much higher level of presnetation. The above two method depend on the prediction of duration and F0 which we know to be a difficult thing to predict (and also there are more than one solution). By not using explcit prosody values into selection prosody will (it is hoped) fall out of the selection process.

7.2.6 Continuity cost

The problem is not just to find a candidate unit which is close to the target unit. It must also be the case that the candidate units we select join together well. Unlike in finding the target cost, this time we have the waveform of the units we are to join, so we can use spectral information in finding the distances. Again after experiment, and some experience we can use

- mel-cepstrum vectors (possibly quantized for efficiency)
- local F0
- local power

Again we can use a weighted sum of differences to provide a *continuity cost*.

We can actually use this measure to not just tell us the cost of joining any two candidate units. We can also use this measure for *optimal coupling* (*conkie96*). This means we can also find the best place to join the two units. We vary the edges within a small amount to find the best place for the join. This technique allows for different joins with different units as well as automatically finding an optimal join. This means the labelling of the database (or diphones) need not be so accurate.

7.2.7 Selecting the units

Given the targets costs, either as weighted distances (*hunt96*) or distances from the cluster centre for

members of the most appropriate cluster (*black97b*), or longer units such as from PSM, and continuity costs the best selection of units from a database is the selection of units which minimises the total cost.

This can be found using Viterbi search with beam pruning.

7.3 Joining the units

After finding the best set of candidate units, we need to join the units. This is true for diphones or nphones or unit selection algorithms. We will also need to modify their pitch and duration (independently) to make the selection units match their targets. In the selection-based synthesis approach, as the selected targets may already be very close to the desired prosody sometime you may get away with no modification.

There are a number of techniques available to do modify pitch and duration with minimum distortion to the signal. Most of these techniques are *pitch synchronous*. That is they view the speech as a stream of short signals starting at the "burst" of a pitch mark to just before the next. In unvoiced speech some arbitrary section is chosen.

To modify duration without modifying pitch we delete (or duplicate) the pitch signals keeping the same distance between them

To modify the pitch we move the signals so they overlap (increase pitch) or become further apart (lower pitch). Usually the signal is windowed (hamming or similar) is done to reduce distortion at the edge.

The following techniques exist in Festival (one is in an external program)

Dumb
: Simply put the selected waveforms together with no modification.

PSOLA
: Pitch-Synchronous Overlap and Add (*moulines90*). This technique works in the time-domain and hence is computationally inexpensive. A patent on the technique is claimed by France Telecom.

Residual/Spike excited LPC
: With LPC parameters to represent the pitch period *hunt89*. Generate a spike of sufficient energy to get reasonable synthesis, or use the original residual to get high quality re-synthesis. This is computationally expensive.

MBROLA
: Multi-band overlap and add *dutoit93*.

7.4 Exercises

1. Using the various parts you've built during the other exercises, built a voice which sounds like a synthesizer. Intonation is probably the key here, monotone with rise at end of utterance for questions and falls for others. Modify the duration to be fixed (`Parameter.set 'Duration_Method 'Default`). Its important that the voice sound synthetic rather than bad.
2. Build a French synthesizer, using the MBROLA French diphone database.

7.5 Hints

1. Use the previous examples, and *listen* to the result and modify it as you see fit. Ask about things you don't know how to "down grade"
2. A start at a French synthesizer is in `` `COURSEDIR/scm/french_mbrola.scm' `` following the directions described in the Festival manual itself in the section on "Building a new voice." This is a lot of work so its not necessary to do this, but at least try and get it to say hello.

   ```
   (voice_french_mbrola)
   (SayText "bonjour")
   ```

   The above will work with no changes. Alternatively try to build a full text to speech synthesizer for Klingon. We have a set of Klingon diphones and you can probably find more information by search for "Klingon Language" on the web.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

8 Building models from databases

Festival is intended to be a framework within which you can experiment with new synthesis techniques. One of the major directions we are going is the automatic training of models from data in databases of natural speech. This chapter shows some of the functions Festival supports to make data extraction and model building from databases easy.

This chapter is split into four sections, collecting databases, labelling, extraction of features and building models from the data.

8.1 Collecting databases

Getting the right type of data is important when we are going to be using its content to build models. Future work in Festival is likely to allow building of waveform synthesizers from databases of natural speech from a single speaker. These unit selection techniques encapture the properties of the database itself so it is important that the database is of the right form. A database of isolated words may provide clearly articulated synthesized speech but it is going to sound like isolated words, even when used for the synthesis of continuous speech. Therefore if you wish your synthesizer to read the news stories it is better if your database contains the same sort of data, i.e. the reading of news stories.

It is not just the acoustic properties of the database that are important, the prosodic properties are important too. If you are going to use your synthesizer to synthesize a wide range of intonational tunes, your database should have a reasonable number of examples of these if you wish any model to be properly trained from the data. Although prosodic rules may be hand written, or trained models hand-modified to cover phenomena not actually present or rare in your database (e.g. phrase final rises), if unit selection synthesis is being used your prosodic models will be predicting variation that is not present in the database and hence unit selection will not be optimal. Thus in unit selection synthesis prosodic models should, where possible, be trained (or at least parameterized) from the same database that is used for unit selection.

The following specific points are worth considering when designing a database.

phonetic coverage
: This is probably the most important criteria. Note that its not quite as simple as it sounds. Some phonemes are particularly rare, especially when they become effectively optional when labelling. For example syllabic l, m and n, may or may not be used depending on the phoneme set selected.

Often syllabic consonants are included in the phone set but they are only occasionally labelled in the database itself. Their rarity makes it difficult to resynthesize them. Of course you'll need at least one occurrence of every phone in you database but many more in different contexts is desirably. Note that there is a distinction between *phonetically balanced* and *phonetically rich* databases. The first offers phones in the approximate distribution they appear in normal speech while the second biases distribution more towards the rarer combinations. Diphone databases typically go for *phonetically rich* in that they ensure occurrences of each phone explicitly in *all* (left) contexts. This measure is probably too strict for unit selection type databases and going for *phonetically balanced* but with a reasonable number of the least common phones is probably better as it offers more natural speech.

dialect

It is useful to have your speaker's dialect match your lexicon, as lexicons are difficult to reliably alter. This effectively limits databases to major dialects. Hopefully this will change with more generalized lexicons but at present better results will be attained on more standard dialects.

voice quality

Selecting a "good" speaker will make your synthesis better. "Good" unfortunately cannot yet be fully characterized. However many of the qualities people associate with good, clear speakers are exactly the properties that make it easier to do unit selection, and build prosodic models. Consistency is probably the most important characteristic. Fast, unusually widely varying speech is going to be harder to capture with models than clear consistent speech. That is, a synthesizer build from Patrick Stewart's voice is likely to be better than one built from Jim Carey's. Realistically be careful not to set your sights too high and hope to synthesize "character" voices. They are probably still too difficult to properly model. It is a lot of work to record and label a database so think carefully about the speaker's voice before you embark on the task.

prosodic coverage

Database consisting of just isolated words will contain inherent properties of isolated spoken words no matter what tricks you apply to your model building. Durations are typically longer, phones are more articulated and there are less intonational tunes. Sentences are better but they will still lack varying prosody that is used when reading longer passages. Isolated sentences (e.g. like TIMIT 460) lack interesting prosodic phrasing, and little intonation variation (typically no continuation rises, emphasis or phrase final rises). Reading short paragraphs is closer to the ideal, including quoted dialogue will also help the variation. Also common phrases, greetings often have their own very subtle properties that are very difficult to synthesize. Typically synthesis of "hello" is never as good as "The quick brown fox jumped over the lazy dog." Including common greetings (in proper context) are therefore worthwhile if you wish to synthesize such forms.

size

Obviously the larger the size of the database the better, but only within reason. The larger the size the more time would be needed to record it. Given multiple recording sessions the quality could be different making it less useful for unit selection. However such recording differences are less important for building prosodic models so more data can only help (colds, hangovers, etc. notwithstanding). Also remember that the more data you get the more you will need to label. A minimum size for unit selection is probably something like the 200 CSTR phonetically balanced sentences. These consist of about 10 minutes of speech (about 9,000 phones). This is really too

small to train durations and intonation models from, though they can be used to parameterize models trained on larger data sets. The TIMIT 460 phonetically balanced sentences is probably a more acceptable option (around 15,000 phones). We are likely to use this as a base database size for at least some of our future experiments, because copies already exist and it is a clearly defined set that others can access to record their own databases. As mentioned above it does have prosodic limitations, but for the time being we can live with these. Better prosodic coverage may be found in databases like Boston University's Radio News Corpus *ostendorf95*. These are single speaker databases of short news stories. The f2b database consists of about 45 minutes of speech (about 40,000 phones), while f3a consists of almost 120 minutes (about 100,000 phones). The text used to build these databases is unfortunately under copyright so you will need to find your own text to use. Depending on future developments in Festival, we hope to release explicit recommendations for sizes and content of databases and their relative trade-offs

Ideally the database should be tuned for the requirements of the synthesizer voice. It therefore should have

- Phonetic coverage, by careful design or be large enough (e.g. 45 minutes of speech). Specific additions may be added to cover unusual phone sequences.
- Utterances should contain paragraphs rather than (in addition to) isolated sentences.
- If the system is to be used for dialog, dialog examples should be included.
- It is wise to include a number of occurrences of phrases, words that are most likely to be used. If the system is going to be used to give out telephone numbers, it is wise to actually include a reasonable number of number examples in the database. Also common greetings often have quite subtle prosodic qualities that people are aware of. Introductions and standard greetings should also be included, this will aid naturalness.

Over all, around an hour of speech should be sufficient. Pruning methods are likely to allow reduction of this for unit selection but the whole database can be used for training of prosodic models.

Many of the techniques used to join and manipulate the units selected from a database require pitch marking, so it is best if a larynograph is used during the recording session. A larynograph records impedance across the vocal folds. This can be used to more accurately find the pitch marks within a signal. A head mounted microphone should also be used. Because of the amount of resources required to record, label and tune a database to build a speech waveform synthesizer, care should be taken to record the highest quality signal is the best surroundings. We are still some way from getting people to talk into a cheap far-field microphone on their PC while the TV plays in the background and then successfully build a high quality synthesizer from it.

8.2 Labelling databases

In order for Festival to use a database it is most useful to build utterance structures for each utterance in the database. As we talked about earlier utterance structures contain streams of ordered items, with

relations between these items. Given such a structure we can easily read in the full utterance structure and access it, dumping information in a normalised way allowing for easy building and testing of models.

Of course the level of labelling that exists, or that you are willing to do by hand or using some automatic tool for a particular database will vary. For many purposes you will at least need phonetic labelling. Hand labelled data is still better than auto-labelled data, but that could change. The size and consistency of the data is important too, though further issues regarding that subject are dealt with in the next chapter.

In all for this example we will need labels for: segments, syllables, words, phrases, intonation events, pitch targets. Some of these can be derived, some need to be labelled.

These files are assumed to be stored in a directory `festival/relations/' with a file extension identifying which utterance relation they represent. They should be in Entropic's Xlabel format, though its fairly easy to convert any reasonable format to the Xlabel format.

Segment
: These give phoneme labels for files. Note the these labels *must* be members of the phoneset that you will be using for this database. Often phone label files may contain extra labels (e.g. beginning and end silence) which are not really part of the phoneset. You should remove (or relabel) these phones accordingly.

Word
: Again these will need to be provided. The end of the word should come at the last phone in the word (or just after). Pauses/silences should not be part of the word.

Syllable
: There is a chance these can be automatically generated from, Word and Segment files given a lexicon. Ideally these should include lexical stress.

IntEvent
: These should ideally mark accent/boundary tone type for each syllable, but this almost definitely require hand-labelling. Also given that hand-labelled of accent type is often not very accurate, its arguable that anything other than accented vs. non-accented can be used reliably.

Phrase
: This could just mark the last non-silence phone in each utterance, or before any silence phones in the whole utterance. The script, `make_Phrase' marks a phrase break before all non-continuous silences and at the the end of utterances based on the Segment files.

Target
: This can be automatically derived from an F0 file and the Segment files. A marking of the mean F0 in each voiced phone seem to give adequate results. The script `make_Target' will do this assuming Segment files, and F0 files. A slight modification to this file will also generate F0 files using `pda' which is included with the Edinburgh Speech Tools or ESPS's `get_f0'. Better results may easily be possible by using more appropriate parameters to pitch extraction program.

Once these files are created a utterance file can be automatically created from the above data. Note it is

pretty easy to get the simply list relations right but building the relations between these simple lists is a little harder. Firstly labelling is rarely accurate and small windows of error must be allowed to ensure things line up properly. The second problem is that some label files identify point type information (IntEvent and Target) while others identify segments (e.g. Segment, Words etc.). Relations have to know this in order to get it right. For example it not right for all syllables between two IntEvents to be linked to the latter IntEvent, only to the Syllable the last IntEvent is within.

The script `make_utts' automatically builds the utterance files from the above labelled files.

This script will generate utterance files for each example file in the database which can be loaded into Festival and used either to do "natural synthesis", or used to extract data for training or test data for building models.

8.3 Extracting features

The easiest way to extract features form a labelled database of the form described in the previous section is by loading in each of the utterance structures and dumping the desired features.

Using the same mechanism to extract the features as will eventually be used by models built from the features has the important advantage of avoiding spurious errors easily introduced when collecting data. For example a feature such as `n.accent` in a Festival utterance will be defined as 0 when there is no next accent. Extracting all the accents and using an external program to calculate the next accent may make a different decision so that when the generated model is used a different value for this feature will be produced. Such mismatches in training models and actual use are unfortunately common, so using the same mechanism to extract data for training, and for actual use it worthwhile.

The Festival function `utt.features` takes an utterance, a relation name and a list of desired features as an argument. This function can be used to dump desired features for each item in a desired relation in each utterance.

The script `festival/examples/dumpfeats' gives Scheme code to dump features. Its takes argument identifying the Relation to be dumped, the desired features, and the utterances to dump them from. The resulst may be dumped into a single file or into a set of files based on the name of the utterance. Also arbitrary other code may be included to add new features.

8.4 Building models

This section describes how to build models from data extracted from databases as described in the previous section. It uses the CART building program, `wagon' which is available as with the Edinburgh Speech Tools Library. But the data is suitable for many other types of model building

techniques, such as linear regression or neural networks.

Wagon is described in the Speech Tools Library Manual, though we will cover simple use here. To use Wagon you need a datafile and a data description file.

A datafile consists of a number of vectors one per line each containing the same number of fields. This, not coincidentally, is exactly the format produced by the `dumpfeats' described in the previous section. The data description file describes the fields in the datafile and their range. Fields may be of any of the following types: class (a list of symbols), floats, or ignored. Wagon will build a classification tree if the first field (the predictee) is of type class, or a regression tree if the first field is a float. An example data description file would be

```
(
( segment_duration float )
( name # @ @@ a aa ai au b ch d dh e e@ ei f g h i i@ ii jh k l m n
    ng o oi oo ou p r s sh t th u u@ uh uu v w y z zh )
( n.name # @ @@ a aa ai au b ch d dh e e@ ei f g h i i@ ii jh k l m n
    ng o oi oo ou p r s sh t th u u@ uh uu v w y z zh )
( p.name # @ @@ a aa ai au b ch d dh e e@ ei f g h i i@ ii jh k l m n
    ng o oi oo ou p r s sh t th u u@ uh uu v w y z zh )
( R:SylStructure.parent.position_type 0 final initial mid single )
( pos_in_syl float )
( syl_initial 0 1 )
( syl_final 0 1)
( R:Sylstructure.parent.R:Syllable.p.syl_break 0 1 3 )
( R:Sylstructure.parent.syl_break 0 1 3 4 )
( R:Sylstructure.parent.R:Syllable.n.syl_break 0 1 3 4 )
( R:Sylstructure.parent.R:Syllable.p.stress 0 1 )
( R:Sylstructure.parent.stress 0 1 )
( R:Sylstructure.parent.R:Syllable.n.stress 0 1 )
)
```

The script `COURSEDIR/bin/make_wgn_desc' goes some way to helping you build a Wagon description file. Given a datafile and a file contain the field names, it will construct an approximation of the description file. This file should still be edited as all fields are treated as of type class by `make_wgn_desc' and you may want to change them some of them to float.

The data file must be a single file, although we created a number of feature files by the process described in the previous section. From a list of file ids select, say, 80% of them, as training data and cat them into a single datafile. The remaining 20% may be catted together as test data.

To build a tree use a command like

```
wagon -desc DESCFILE -data TRAINFILE -test TESTFILE
```

The minimum cluster size (default 50) may be reduced using the command line option `-stop` plus a number.

Varying the features and stop size may improve the results.

Also you can try `-stepwise` which will look for the best features incrementally, testing the result on the test set

Building the models and getting good figures is only one part of the process, you must integrate this model into Festival is its going to be of any use. In the case of CART trees generated by Wagon, Festival supports these directly. In the case of CART trees predicting zscores, or factors to modify duration averages by such trees can be used as is.

Other parts of the distributed system use CART trees, and linear regression models that were training using the processes described in this chapter. Some other parts of the distributed system use CART trees which were written by hand and may be improved by properly applying these processes.

8.5 Exercises

These exercises ideally require a labelled database, but an example one is provided in `` `COURSEDIR/gsw/' `` to make the exercises possible without requiring hours of phonetic labelling.

1. From the given Segment, Word, Syllable and IntEvent files, create the Phrase and Target files. Then generate your own version of the utterance files. If you have an alternative database try and build as many of these files as you can and build utterances files for each file in your database.
2. Either for your own database or the from the utterances provided, extract data from which we will build duration models. As a start extract the following features

```
segment_duration
name
p.name
n.name
R:SylStructure.parent.position_type
pos_in_syl
syl_initial
syl_final
R:SylStructure.parent.R:Syllable.p.syl_break
R:SylStructure.parent.syl_break
R:SylStructure.parent.R:Syllable.n.syl_break
```

```
R:SylStructure.parent.R:Syllable.p.stress
R:SylStructure.parent.stress
R:SylStructure.parent.R:Syllable.n.stress
```

for each segment in the database. Split this data into train and test data and build a CART tree from the training data and test it against the test data. Add any other features you think useful (especially the `ph_` features), to see if you can get better results.

3. Try the same prediction again but use zscores, and/or log durations and see what the relative results are.

8.6 Hints

1. For new databases, this is not necessarily easy and may require quite a lot of hand checking, but the results are worth it.
2. You will need to use `wagon` for this. `wagon` requires a description file and a datafile. The data file is as dumped by `dumpfeats`. The description file must be written by hand, or use `` `speech_tools/bin/make_wgn_desc' `` to give an approximation. To run `` `wagon' `` use a command like

   ```
   wagon -desc DESCFILE -data TRAINDATA -test TESTDATA
   ```

   The default minimum cluster size is 50, try to vary that using the the option `-stop` to get better results.
3. You will have to write simple Unix functions to collect averages and calculate zscores for each feature vector in your datafile. Remember the RMSE and correlation are fundamentally different in different domains so cannot be directly compared. You would need to convert the data back into the same domain to compare to different prediction methods properly.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

9 New Voices

The FestVox projects (http://festvox.org) specifically deals with this issue and contains full documentation, disucssion, as wellas tools for designing, recording, building, and testing of new voices in new and currentl supported languages. If you are going to actual build a new synthetic voice you should read the festvox document itself

9.1 Building New Voices

Building a new voice in a new language requires addressing the following

- Phoneset
- Token processing rules
- Prosodic phrasing method
- Word pronunciation (lexicon and/or letter to sound rules)
- Intonation (accents and F0 contour)
- Duration
- Waveform synthesizer

Before starting you must have a phoneset. This is often pre-defined due to the lexicon and/or waveform synthesizer that is to be used. Most languages have a phoneste already defined for them, though if this is a language that has had little computational the phoneset may not be defined complete for easy computational use. However wherever possible it is wise to use someone elses defined phoneste but do consider the possibility that it needs to be augment due to allophonic phenomena such as vowel reduction, flaps, palatalization etc.

The next area to consider is tokenization of text, assuming the desired language has a written form, if not you'll need to defined on. This can be relatively easy or not depending on the language. Chinese, Japanese and Thai (and others) do not use spaces between works so even simple tokenization is a non-trival process. There are statistical techniques that work for this but they require a lexicon of words in the language *sproat96b*. Other problems such as numbers, symbols, homographs etc require treatment. Note that although from a anglosized view treatment of text in the native writting system may seem no worth it, for it to be considered a TTS system for that language you must allow input to be in that writing system. This requirement may be non-trivial to answer if there are few computer fonts, or authoring tools for that language, but it should be seriously addressed.

In some cases a romanized only form of synthesis may be sufficient, for example if the synthesizer is only going to be used in language generation systems, or manchine translation system, it *may* be possible to use this simplified script. Also note that due to the dominence of ASCII based machines that even when there is a native written form many computer based users of the language have constructed a romanized system due to the inadequacy of existin computer systems. Support that writting system may be advanatageous as there already exist text (e.g. email) in that form. For example Greek can be written in Ascii using soundalike and lookalike symbols. Greek people do use this in email when no other input method is available and we found it useful to include support for this in our Greek synthesizer.

Possibily the most expensive part required for a synthesizer is probably a lexicon. However many language don't require one as the relationship between the written and phonetic form can be so close that simple hand written letter to sound rules are sufficient. Building large lexicons is a long and tedious task and not something to be undertaken lightly. Whereever possible you should try to find an existing lexicon in the language. Though remember to note its copyright restrictions.

Note that for some languages, which have a rich morphology such as Finnish and Turkish, its not easy to list the words in the language due to the large number of variations in forms. Even in langauge with relatively few variations it is impossible to list every word in the language, new words, rare words, especially proper and place names will appear in small but regular frequency in new text to be spoken and some pronunciation is will be required. As there is often some, even though non-obvious, relation between the written form and the pronunciation trainable letter to sound rules systems such as support by festival are a solution. However there may always be symbols that cannot be pronounced. Though in this case we should defer to what a human would do. FOr example what does a Chinese or Japanese person do when they come accross a chinese character they have never seen before, or for that matter an English speaking person when they come across a symbol that denotes an artist formerly known as Prince.

9.2 Limited domain synthesis

Unit selection synthesis offers high quality natural sound utterances, but unfortunately it can also deliver quite incomprehensible synthesis. Most of the work in unit selection synthesis is invovled in trying to automaticall detect when its gone wrong and try to ensure this happens as infrequently as possible. The reasons for the bad quality examples are first, that the acoustic measures and optimal coupling are not simple reliable measures of quality, and hence the search for better measures that better reflect perceptual notions of good quality speech. The second reaosn is that the database being selected from doesn't have the distribution of units that are required for synthesizing the desired utterance. It is obvious when using a unit seleciton synthesizer that it works better when there are more example of what you wish in the database.

To try to take advantage of this second point in producing high quality synthesis more reliably we can build synthesizer from database that are deliberated tailored to the task they are to perform. At the most extreme end we could simply record all the utterances that are required for the particular task we require a synthesizer for. But that seems over restrictive and possible a lot of recording note that a task may still

have a restricted doamin even through it has an infinite number of utterances.

For example consider a talking clock. We can specify a particular structure for the how to say the time, and the build a database that has a resonable number of examples of the munites and hours in each slot. For example included in the Festival distribution is a simple `saytime' script which gives the current time in a simple approximate form as in

```
The time is now, a little after ten past eleven, in the morning.
```

Thus we can design a number of utterances which cover all the basic word/phrases that such an program needs to say. In this case 24 utterances seems to get the coverage we need. These are

```
"The time is now, exactly five past one, in the morning."
"The time is now, just after ten past two, in the morning."
"The time is now, a little after quarter past three, in the morning."
"The time is now, almost twenty past four, in the morning."
"The time is now, exactly twenty-five past five, in the morning."
"The time is now, just after half past six, in the morning."
"The time is now, a little after twenty-five to seven, in the morning."
"The time is now, almost twenty to eight, in the morning."
"The time is now, exactly quarter to nine, in the morning."
"The time is now, just after ten to ten, in the morning."
"The time is now, a little after five to eleven, in the morning."
"The time is now, almost twelve."
"The time is now, just after five to one, in the afternoon."
"The time is now, a little after ten to two, in the afternoon."
"The time is now, exactly quarter to three, in the afternoon."
"The time is now, almost twenty to four, in the afternoon."
"The time is now, just after twenty-five to five, in the afternoon."
"The time is now, a little after half past six, in the evening."
"The time is now, exactly twenty-five past seven, in the evening."
"The time is now, almost twenty past eight, in the evening."
"The time is now, just after quarter past nine, in the evening."
"The time is now, almost ten past ten, in the evening."
"The time is now, exactly five past eleven, in the evening."
"The time is now, a little after quarter to midnight."
```

We then record these and build a unit selection synthesizer from them. This synthesizer can *only* say things in domain. As the above doesn't even have phoneme coverage for English it is unquestionably a limited domain synthesizer, but when it talks the quality is far superior to diphones and the quality is almost never poor, even though there are still some problems with it.

http://festvox.org/ldom/ldom_time.html includes so examples of a talking clock using this technique. And

the festvox documentation include detailed instructions on building such a talking clock.

However this above is very simple and really too limited. This technique however does scale up, at least to much more interesting domains. A weather reporter include times, dates, weather outlok etc is also available at http://festvox.org/ldom/ldom_weather.html. This was done with 100 sentences in domain of the form

```
The weather at 10:00 am on Sunday April 23, outlook cloudy,
45 degrees, winds northwest 10 mile per hour
```

We have experimented with these techniques futher with a limited domain synthesizer for the CMU Darpa Communicator project, which offers a telpehone dialog system for booking flights etc. Here the output forms seems more general that a simple slot and filler approach but still much is actually constrained. We have built a 500 utterance database which covers most of what is required, flight numbers, times, cities and airlines are the only major changing parts, while the standard expression remain the same (except when we update the systems).

Tailoring your synthesizer to the particular task is always a good idea. But it is also possible a risk. If something needs to be changed in your system, having to re-record is not as easy as simply adding another line of text in your program. Thus there are disadvantages to limited domain synthesizers over general ones.

9.3 Voice Conversion

Instead of carefully collecting a diphone database from a speaker or a balanced database for unit selection it would be much better if only a small sample of speech was necessary from a speaker in order to model their speech. Ideally one would use an existing database of speech and prosodic models and convert them based on a small sample from the desired speaker.

Such techniques are being studied and probably eventually will be the standard method for builing new synthetic voices. Such a technique would reuqire much less participation from the target speaker, include less high quality recordings and less effort on their part.

Apart from the obvious advantage of being able to build a new voice more easily, voice conversion has other advantages too. First it would be easier to have a synthesizer have many more voices available allowing a user to choise if not their own voice on they prefer for their system. If the modelling is done appropriately this can be done without requiring large amounts of disk requirements to store the multiple voices. A second advantage is that modelling might be done across language thus a speech to speech transaltion system can get a voice like the originator but speaking in a language that original person cannot speak. Also imaging a network based telephone system that is bandwidth constrained (i.e. all telephone systems), the modification parameters alone could be tranfered to the other site and a

synthesizer voice from the orignal could be constructed from a very low band width line that passed only phones, durations and F0 points.

A person's voice is a combination of thier lexical, prosodic, and spectral properoties. Thus ideally to model a voice you need sufficient samples to augment existing representations. In some sense we have already discussed dialects and prosodic modelling. Here we will concentrate on the lower level parts.

Given the form of parameterization of the speech we are doing we can view the our standard parameterization to be spectral plus residual. Where the spectral information is contained within the LPC (or cepstrum) parameters. Mapping these parameters to a new voices is the basic technique. Though whwat parameterization, how mapping is done, and how to use the available parameters from the target speaker are all active areas of research.

Oregon Graduate Institute have a number of example of voice conversion on line at http://cslu.cse.ogi.edu/tts/research/vc/vc.html. They also have developed a Festival plugin for this technique *kain98*.

They can achieve reasonable mapping of a male to a female voice from 64 diphones in the target voice. However although they (and others in this area) do definitely achieve a new synthetic voice so far none of the technique really fully capture all the propties of the target voice. Though this is also true when full diphones are collected so it is not just the conversion that is the problem it is the fact that we still do no yet have an adequate model for the repreesentation of a person's voice.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

10 Future directions

This chapter discusses processes in synthesis that are now ripe for detailed research work. The improvement offered by diphone technology and the corresponding jump in quality offered by general unit selection appear to offer significantly higher quality synthesis than anything we have had before. Improvements in speech recognition, speech coding, and of course, larger faster machines make experiments, databases and training algorithms that were once beyond our reach, now practical on reasonably sized systems.

10.1 Synthesis of any voice

One goal of Festival over the next few years is to provide a system that can automatically build high quality new voices for anyone who wants them.

To add a new synthesis voice, the process should be

- Present a number of sentences/utterances to a person and record them saying them.
- Automatically label the recorded data with phones, syllables, words, and intonation.
- Extract parameters for intonation, phrasing, duration, dialectal phone mappings etc. from the labelled data.
- Build a voice database from the extracted parameters and phone labelled data.
- This should be possible for non-linguistic/phonetic trained people and be possible in reasonably time on standard workstations

Festival is still some way from this goal but already the beginnings are there. Note we are not asking for a system that does acquisition of arbitrary languages. This system would be constrained. Although Festival would offer ways to build support of new languages, that process would still require greater skill. Adding a new voice in a supported language, but with possibly different dialects, is our more modest goal.

10.2 Size of database

So what is the best size and type of database to use for such a model? There are two possible views

1. A database large enough to cover enough phonetic and prosodic variation in the language so that

most of the units required are contained within the database and their are sufficient examples of prosodic variations to make the models learn from them.
2. Record only a small amount of speech and use the information in that to modify existing parametric systems. The work on voice conversion would be relevant here. Also as it will be prohibitive to record very large databases, extracting key examples to tune existing models that were trained on larger databases is probably more efficient.

Our current estimate is around 45 minutes of speech which should contain the sort of speech that you wish synthesized. Short paragraphs containing rich phonetic variation are reasonable but if the voice is to be used for reading numbers, or answering the telephone, the database should included numbers, dates, welcome messages (the company name) in various prosodic contexts.

There is unlikely to be one specific text set that is good for all, and different sets will be good for different applications. Also many people will be unwilling to talk to their computer for 45 minutes before they can get their new voice, so multiple levels and sizes of text to say to best optimize build time, quality etc. should be identified.

There are many experiments and design choices required here to come up with detailed guides and database texts that are appropriate when building new voices.

10.3 Autolabelling

Given a database of speech, even with the orthography we will need some way to label this automatically with lower level labels such as phones. Speech recognition technology has improved greatly and using packages like HTK, it appears easy to build an autolabelling system that does reasonably well. The problem however with labelling for synthesis rather than labelling for recognition, is that the labels need to be more accurate for a synthesis database than for word recognition. But its not all bad, as for recognition you (probably) need to recognize all words but for synthesis you need only label the parts you can do so well. Ignoring badly labelled parts is acceptable, as long as there is enough redundancy in the database.

Therefore what we need is:

- a robust phone labelling system
- some measure of correctness for labels

The phone labels need to overlap with examples of phones in the waveform but phone boundaries probably don't need to be very accurate given that the synthesis method will use some form of optimal coupling.

However gross errors need to be identified from mis-alignment, I have seen syllables misaligned in

autolabelled corpora and also occasions when the speaker has actually said something different from the orthography. Some *automatic* measure of accuracy is necessary to judge when to ignore a section, or even ask for re-recording.

Another method is to ignore traditional phonetic classes as labels and use some other system for labelling. As the task of unit selection is to find appropriate units it is reasonable to question the use traditional phonetic classes. Two possible alternatives are mentioned here but others could also be reasonable candidates.

bacchiani96 derives acoustically similar segmental units from coded parameters for speech recognition. Some experiments in speech synthesis have already been done with regards labelling databases for synthesis and they indicate further work could be fruitful, particularly in that this method is accurate at labelling the data, and fully automatic. But of course you still need to match traditional phonetic classes (from the targets) to these acoustically derived labels, but that might be an easier task than mapping to phonetic labels to waveform segments directly.

A second alternative in looking for non-traditional phonetic labelling is described in *fujimura93*. The C/D model offers a multi-layered (autosegmental) labelling system for different phonetic phenomena, it offers a more reasonable labelling for variable speed speech and articulatory constraints in forming speech are implicit in the model.

In addition to phone (or equivalent) labels other forms of labelling are desirable: syllable and word labelling are probably trivial given orthography and phones. Phrasing, and intonation do require further labelling expertise. *taylor94b* provides a method for autolabelling but that has not been fully implemented and measured within a text to speech framework.

It seems reasonable to prompt users when recording a database. When given a prompt a user will most likely mimic the prosody of the prompt itself so this may help seed the intonation labelling system. However as full paragraphs will also be given, which is necessary to get fully fluent speech, users will not remember the full prosodic structure of what they have to say and hence will deviate from the prompt. It may be reasonable to record small sentences with specific variation to seed the intonation labelling system then use that to train labelling for the rest of the corpus.

10.4 Training from labels

Given our neatly labelled database we must use this information sensibly to extract parameters for our various synthesis models

- Lexicons need to be dialect sensitive. If Festival supports English it should trivially support major dialects of it without requiring whole new lexicons or whole new intonation methods. Lexicons would need to be constructed in a way that allow such mapping. Also note that few people speak

in pure dialect and it is the combination of "dialects" that make their speech their own. Their lexicon, intonation parameters etc. should reflect that.

- We need a good *acoustic measure* no matter what eventual form of unit selection we choose, a good automatic measure of how good our synthesis is is necessary for any type of training method. This probably requires a significant psycho-linguistic experiment to find how human perception perceives different selection candidates and then find an automatic measure which follows those perceptions.
- We still do not know which acoustic/phonetic/prosodic features are the most important in any form of unit selection and what their relative weighting (in different contexts) are. Also how adequately various failures in selection can be corrected by signal processing techniques.
- Intonation, duration and other prosodic features still require more work to ensure they are fully automatic.
- Post-lexical rules: how a speaker modifies their pronunciation from a base leixcal form can also be learned. Vowel reduction is is prime example.
- The influence of dialog structure on intonation (questions, agreement focus, topic shifting etc.) is still an open area. With the use of synthesizers in dialog systems a more responsive voice is required making more demands on the intonational properties of the synthesized voice. A CNN newsreader voice is probably not appropriate as a dialog partner when asking for train time-table information.

Go to the first, previous, next, last section, table of contents.

Go to the first, previous, next, last section, table of contents.

---

11 Final comments

This course has covered a number of specific tasks in the process of speech synthesis. The four major areas are

1. architecture
2. text processing
3. linguistic/prosodic processing
4. waveform synthesis

Each was covered in terms of their implementation within Festival.

Synthesis is defined for *utterances* which can be created from simple text or tokens (or whatever) and are filled out through *modules*. The modules fall into the three major processes of synthesis.

Text to speech is defined as the problem of tokenizing arbitrary text, chunking it into utterances and them applying the appropriate modules to render it as waveforms and play it.

Festival offers a flexible architecture where each of the parts may be fully parameterized and controlled allowing the desired effect.

Although the architecture of Festival may seem daunting at first, I hope you now see why it was designed as such so that it can be flexible enough for what ever we need to do.

Note that Festival is still being developed there are many things it will do in the future that it does not yet do. Some of these future enhancements are discussed above but others we will only find their necessity as we implement more parts of the system. Research and development cannot be mapped out fully and must be sensitive to problems found on the way.

These notes only give a brief overview of aspects of speech synthesis for further reading the following texts are recommend.

- "An introduction to Text-to_Speech Synthesis" by Thierry Dutoit, Kluwer Academic Publishers 1997. This contains the most up to date description of the whole text to speech process.
- "Test-to-speech: The MITalk system" by J Allen, M Hunnicut and D Klatt, Cambridge University Press. 1987. This book is a little old now but contains many details that people in speech synthesis

should know about.

- "Multilingual Text-to-Speech Synthesis: The Bell Labs Approach." eds Richard Sproat. This includes a detailed look at the Bell Labs system covering many aspects of the synthesis process and how they are solved within their system. It is a useful insight to many of the research aspects of the TTS process.
- "Talking Machines" eds G. Bailly and C. Benoit. Horth-Holland, 1992. This book is a collection of papers from an ESCA Workshop in Speech Synthesis in Autrans 1990. It contains many useful papers about current synthesis research.
- "Progress in Speech Synthesis" eds. J van Santen, R Sproat, J Olive and J Hirschberg. Springer Verlag 1996. This contains full length papers from the second ESCA Workshop on Speech Synthesis that was held in Mohonk, New York in 1994. Again a good collection of current work in the area.
- There are also a number of conferences where synthesis papers are commonly presented. Eurospeech and ICSLP held every two years (alternately) contain the latest papers and are good places to meet people working in the field.
- The `comp.speech` frequently asked questions (FAQ) by Andrew Hunt is always a good source of links ot information about synthesis and speech technology in general. It is regular updated. It can be found at any of the following addresses as well as being regularly posted to the USENET newsgroup `comp.speech`

  ```
  Australia: http://www.speech.su.oz.au/comp.speech/
  UK: http://svr-www.eng.cam.ac.uk/comp.speech/
  Japan: http://www.itl.atr.co.jp/comp.speech/
  USA: http://www.speech.cs.cmu.edu/comp.speech/
  ```

  Finally you may follow the development of the Festival Speech Synthesis System through its home page

  ```
  http://www.cstr.ed.ac.uk/projects/festival.html
  ```

Go to the first, previous, next, last section, table of contents.