Chapter 8
Speech Synthesis

And computers are getting smarter all the time: Scientists tell us that soon they will be able to talk to us. (By 'they' I mean 'computers': I doubt scientists will ever be able to talk to us.)

Dave Barry

In Vienna in 1769, Wolfgang von Kempelen built for the Empress Maria Theresa the famous Mechanical Turk, a chess-playing automaton consisting of a wooden box filled with gears, and a robot mannequin sitting behind the box who played chess by moving pieces with his mechanical arm. The Turk toured Europe and the Americas for decades, defeating Napolean Bonaparte and even playing Charles Babbage. The Mechanical Turk might have been one of the early successes of artificial intelligence if it were not for the fact that it was, alas, a hoax, powered by a human chessplayer hidden inside the box.

What is perhaps less well-known is that von Kempelen, an extraordinarily prolific inventor, also built between 1769 and 1790 what is definitely not a hoax: the first full-sentence speech synthesizer. His device consisted of a bellows to simulate the lungs, a rubber mouthpiece and a nose aperature, a reed to simulate the vocal folds, various whistles for each of the fricatives. and a small auxiliary bellows to provide the puff of air for plosives. By moving levers with both hands, opening and closing various openings, and adjusting the flexible leather 'vocal tract', different consonants and vowels could be produced.

More than two centuries later, we no longer build our speech synthesizers out of wood, leather, and rubber, nor do we need trained human operators. The modern task of **speech synthesis**, also called **text-to-speech** or **TTS**, is to produce speech (acoustic waveforms) from text input.

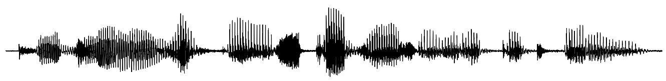
Speech synthesis
Text-to-speech
TTS

Modern speech synthesis has a wide variety of applications. Synthesizers are used, together with speech recognizers, in telephone-based conversational agents that conduct dialogues with people (see Ch. 23). Synthesizer are also important in non-conversational applications that speak **to** people, such as in devices that read out loud for the blind, or in video games or children's toys. Finally, speech synthesis can be used to speak **for** sufferers of neurological disorders, such as astrophysicist Steven Hawking who, having lost the use of his voice due to ALS, speaks by typing to a speech synthesizer and having the synthesizer speak out the words. State of the art systems in speech synthesis can achieve remarkably natural speech for a very wide variety of input situations, although even the best systems still tend to sound wooden and are limited in the voices they use.

The task of speech synthesis is to map a text like the following:

(8.1) `PG&E will file schedules on April 20.`

to a waveform like the following:



Speech synthesis systems perform this mapping in two steps, first converting the input text into a **phonemic internal representation** and then converting this internal representation into a waveform. We will call the first step **text analysis** and the second step **waveform synthesis** (although other names are also used for these steps).

Text analysis
Waveform synthesis

A sample of the internal representation for this sentence is shown in Fig. 8.1. Note that the acronym `PG&E` is expanded into the words P G AND E, the number `20` is expanded into *twentieth*, a phone sequence is given for each of the words, and there is also prosodic and phrasing information (the *'s) which we will define later.

			*			*			* L-L%
P	G	AND	E	WILL	FILE	SCHEDULES	ON	APRIL	TWENTIETH
p iy	jh iy	ae n d	iy	w ih l	f ay l	s k eh jh ax l z	aa n	ey p r ih l	t w eh n t iy ax th

Figure 8.1 Intermediate output for a unit selection synthesizer for the sentence *PG&E will file schedules on April 20.*. The numbers and acronyms have been expanded, words have been converted into phones, and prosodic features have been assigned.

While text analysis algorithms are relatively standard, there are three widely different paradigms for waveform synthesis: **concatenative synthesis**, **formant synthesis**, and **articulatory synthesis**. The architecture of most modern commercial TTS systems is based on concatenative synthesis, in which samples of speech are chopped up, stored in a database, and combined and reconfigured to create new sentences. Thus we will focus on concatenative synthesis for most of this chapter, although we will briefly introduce formant and articulatory synthesis at the end of the chapter.

Hourglass metaphor

Fig. 8.2 shows the TTS architecture for concatenative unit selection synthesis, using the two-step **hourglass metaphor** of Taylor (2008). In the following sections, we'll examine each of the components in this architecture.

8.1 Text Normalization

text normalization

In order to generate a phonemic internal representation, raw text first needs to be preprocessed or **normalized** in a variety of ways. We'll need to break the input text into sentences, and deal with the idiosyncracies of abbreviations, numbers, and so on. Consider the difficulties in the following text drawn from the Enron corpus (Klimt and Yang, 2004):

> He said the increase in credit limits helped B.C. Hydro achieve record net income of about $1 billion during the year ending March 31. This figure does not include any write-downs that may occur if Powerex determines that any of its customer accounts are not collectible. Cousins, however, was insistent that all debts will be collected: "We continue to pursue monies owing and we expect to be paid for electricity we have sold."

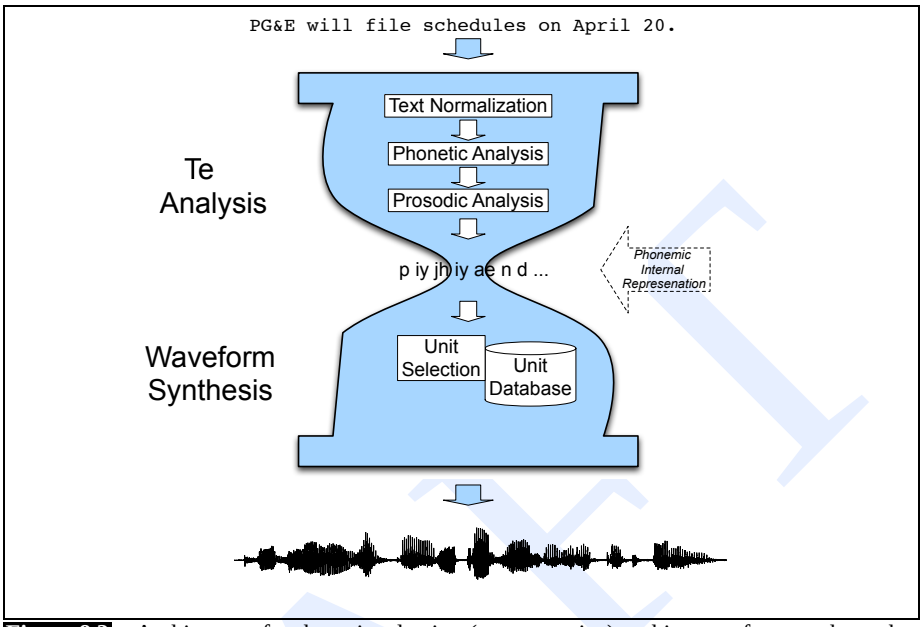


Figure 8.2 Architecture for the unit selection (concatenative) architecture for speech synthesis.

Sentence tokenization

The first task in text normalization is **sentence tokenization**. In order to segment this paragraph into separate utterances for synthesis, we need to know that the first sentence ends at the period after *March 31*, not at the period of *B.C.*. We also need to know that there is a sentence ending at the word *collected*, despite the punctuation being a colon rather than a period. The second normalization task is dealing with **non-standard words**. Non-standard words include number, acronyms, abbreviations, and so on. For example, *March 31* needs to be pronounced *March thirty-first*, not *March three one*; *$1 billion* needs to be pronounced *one billion dollars*, with the word *dollars* appearing after the word *billion*.

8.1.1 Sentence Tokenization

We saw two examples above where sentence tokenization is difficult because sentence boundaries are not always indicated by periods, and can sometimes be indicated by punctuation like colons. An additional problem occurs when an abbreviation ends a sentence, in which case the abbreviation-final period is playing a dual role:

(8.2) He said the increase in credit limits helped B.C. Hydro achieve record net income of about $1 billion during the year ending March 31.

(8.3) Cousins, however, was insistent that all debts will be collected: "We continue to pursue monies owing and we expect to be paid for electricity we have sold."

(8.4) The group included Dr. J. M. Freeman and T. Boone Pickens Jr.

A key part of sentence tokenization is thus period disambiguation; we've seen a simple perl script for period disambiguation in Ch. 3. Most sentence tokenization algorithms are slightly more complex than this deterministic algorithm, and in particular

are trained by machine learning methods rather than being hand-built. We do this by hand-labeling a training set with sentence boundaries, and then using any supervised machine learning method (decision trees, logistic regression, SVM, etc) to train a classifier to mark the sentence boundary decisions.

More specifically, we could start by tokenizing the input text into tokens separated by whitespace, and then select any token containing one of the three characters `!`, `.` or `?` (or possibly also `:`). After hand-labeling a corpus of such tokens, then we train a classifier to make a binary decision (EOS (end-of-sentence) versus not-EOS) on these potential sentence boundary characters inside these tokens.

The success of such a classifier depends on the features that are extracted for the classification. Let's consider some feature templates we might use to disambiguate these **candidate** sentence boundary characters, assuming we have a small amount of training data, labeled for sentence boundaries:

- the prefix (the portion of the candidate token preceding the candidate)
- the suffix (the portion of the candidate token following the candidate)
- whether the prefix or suffix is an abbreviation (from a list)
- the word preceding the candidate
- the word following the candidate
- whether the word preceding the candidate is an abbreviation
- whether the word following the candidate is an abbreviation

Consider the following example:

(8.5) ANLP Corp. chairman Dr. Smith resigned.

Given these feature templates, the feature values for the period `.` in the word `Corp.` in (8.5) would be:

```
PreviousWord = ANLP          NextWord = chairman
Prefix = Corp                Suffix = NULL
PreviousWordAbbreviation = 1 NextWordAbbreviation = 0
```

If our training set is large enough, we can also look for lexical cues about sentence boundaries. For example, certain words may tend to occur sentence-initially, or sentence-finally. We can thus add the following features:

- Probability[candidate occurs at end of sentence]
- Probability[word following candidate occurs at beginning of sentence]

Finally, while most of the above features are relatively language-independent, we can use language-specific features. For example, in English, sentences usually begin with capital letters, suggesting features like the following:

- case of candidate: Upper, Lower, AllCap, Numbers
- case of word following candidate: Upper, Lower, AllCap, Numbers

Similary, we can have specific subclasses of abbreviations, such as honorifics or titles (e.g., Dr., Mr., Gen.), corporate designators (e.g., Corp., Inc.), or month-names (e.g., Jan., Feb.).

Any machine learning method can be applied to train EOS classifiers. Logistic regression and decision trees are two very common methods; logistic regression may

have somewhat higher accuracy, although we have instead shown an example of a decision tree in Fig. 8.3 because it is easier for the reader to see how the features are used.

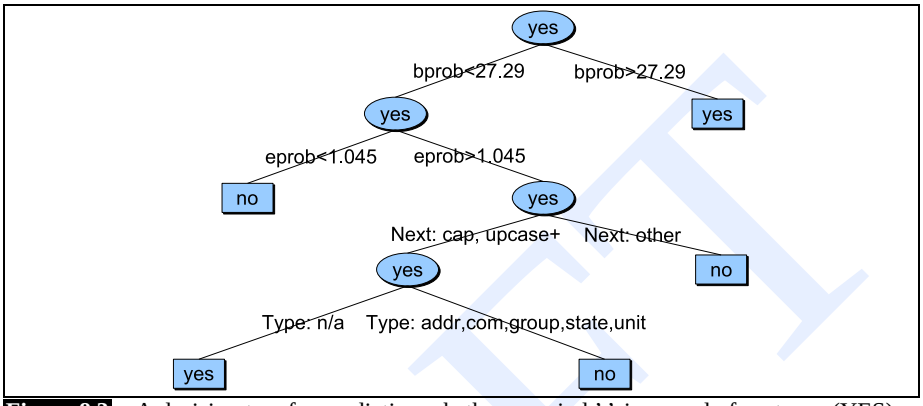


Figure 8.3 A decision tree for predicting whether a period '.' is an end of sentence (YES) or not an end-of-sentence (NO), using features like the log likelihood of the current word being the beginning of a sentence (`bprob`), the previous word being an end of sentence (`eprob`), the capitalization of the next word, and the abbreviation subclass (company, state, unit of measurement). After slides by Richard Sproat.

8.1.2 Non-Standard Words

Non-standard words

The second step in text normalization is normalizing **non-standard words**. Non-standard words are tokens like numbers or abbreviations, which need to be expanded into sequences of English words before they can be pronounced.

What is difficult about these non-standard words is that they are often very ambiguous. For example, the number *1750* can be spoken in at least three different ways, depending on the context:

`seventeen fifty:` (in *'The European economy in 1750'*)
`one seven five zero:` (in *'The password is 1750'*)
`seventeen hundred and fifty:` (in *'1750 dollars'*)
`one thousand, seven hundred, and fifty:` (in *'1750 dollars'*)

Similar ambiguities occur for Roman numerals like *IV*, (which can be pronounced `four`, `fourth`, or as the letters `I V` (meaning 'intravenous')), or *2/3*, which can be `two thirds` or `February third` or `two slash three`.

In addition to numbers, various non-standard words are composed of letters. Three types non-standard words include **abbreviations**, **letter sequences**, and **acronyms**. Abbreviations are generally pronounced by **expanding** them; thus *Jan 1* is pronounced `January first`, and *Wed* is pronounced `Wednesday`. **Letter sequences** like *UN*, *DVD*, *PC*, and *IBM* are pronounced by pronouncing each letter in a sequence (IBM is thus pronounced *ay b iy eh m*). **Acronyms** like *IKEA*, *MoMA*, *NASA*, and *UNICEF* are pronounced as if they were words; *MoMA* is pronounced *m ow m ax*. Ambiguity

occurs here as well; should *Jan* be read as a word (the name `Jan`) or expanded as the month `January`?

These different types of numeric and alphabetic non-standard words can be summarized in Fig. 8.1.2. Each of the types has a particular realization (or realizations). For example, a year NYER is generally read in the **paired** method, in which each pair of digits is pronounced as an integer (e.g., `seventeen fifty` for 1750), while a U.S. zip code NZIP is generally read in the **serial** method, as a sequence of single digits (e.g., `nine four one one zero` for *94110*). The type BMONEY deals with the idiosyncracies of expressions like *$3.2 billion*, which must be read out with the word `dollars` at the end, as `three point two billion dollars`.

Paired digits

Serial digits

For the alphabetic NSWs, we have the class EXPN for abbreviations like *N.Y.* which are expanded, LSEQ for acronyms pronounced as letter sequences, and ASWD for acronyms pronounced as if they were words.

ALPHA	EXPN	abbreviation	*adv, N.Y., mph, gov't*
	LSEQ	letter sequence	*DVD, D.C., PC, UN, IBM,*
	ASWD	read as word	*IKEA*, unknown words/names
NUMBERS	NUM	number (cardinal)	*12, 45, 1/2, 0.6*
	NORD	number (ordinal)	*May 7, 3rd, Bill Gates III*
	NTEL	telephone (or part of)	*212-555-4523*
	NDIG	number as digits	*Room 101*
	NIDE	identifier	*747, 386, I5, pc110, 3A*
	NADDR	number as street address	*747, 386, I5, pc110, 3A*
	NZIP	zip code or PO Box	91020
	NTIME	a (compound) time	*3.20, 11:45*
	NDATE	a (compound) date	*2/28/05, 28/02/05*
	NYER	year(s)	*1998, 80s, 1900s, 2008*
	MONEY	money (US or other)	*$3.45, HK$300, Y20,200, $200K*
	BMONEY	money tr/m/billions	*$3.45 billion*
	PRCT	percentage	*75% 3.4%*

Figure 8.4 Some types of non-standard words in text normalization, selected from Table 1 of Sproat et al. (2001); not listed are types for URLs, emails, and some complex uses of punctuation.

Dealing with non-standard words requires at least three steps: **tokenization** to separate out and identify potential non-standard words, **classification** to label them with a type from Fig. 8.1.2, and **expansion** to convert each type into a string of standard words.

In the tokenization step, we can tokenize the input by whitespace, and then assume that any word which is not in the pronunciation dictionary is a non-standard word. More sophisticated tokenization algorithms would also deal with the fact that some dictionaries already contain some abbreviations. The CMU dictionary, for example, contains abbreviated (and hence incorrect) pronunciations for *st, mr, mrs*, as well as day and month abbreviations like *mon, tues, nov, dec*, etc. Thus in addition to unseen words, we also need to label any of these acronyms and also single-character token as potential non-standard words. Tokenization algorithms also need to split words which are combinations of two tokens, like *2-car* or *RVing*. Words can be split by simple heuristics, such as splitting at dashes, or at changes from lower-case to upper-case.

The next step is assigning a NSW type; many types can be detected with simple regular expressions. For example, NYER could be detected by the following regular expression:

/(1[89][0-9][0-9])|(20[0-9][0-9]/

Other classes might be harder to write rules for, and so a more powerful option is to use a machine learning classifier with many features.

To distinguish between the alphabetic ASWD, LSEQ and EXPN classes, for example we might want features over the component letters. Thus short, all-capital words (*IBM*, *US*) might be LSEQ, longer all-lowercase words with a single-quote (*gov't*, *cap'n*) might be EXPN, and all-capital words with multiple vowels (*NASA*, *IKEA*) might be more likely to be ASWD.

Another very useful features is the identity of neighboring words. Consider ambiguous strings like *3/4*, which can be an NDATE `march third` or a *num* `three-fourths`. NDATE might be preceded by the word *on*, followed by the word *of*, or have the word *Monday* somewhere in the surrounding words. By contrast, NUM examples might be preceded by another number, or followed by words like *mile* and *inch*. Similarly, Roman numerals like *VII* tend to be NORD (*seven*) when preceded by *Chapter*, *part*, or *Act*, but NUM (*seventh*) when the words *king* or *Pope* occur in the neighborhood. These context words can be chosen as features by hand, or can be learned by machine learning techniques like the **decision list** algorithm of Ch. 8.

We can achieve the most power by building a single machine learning classifier which combines all of the above ideas. For example, the NSW classifier of (Sproat et al., 2001) uses 136 features, including letter-based features like *'all-upper-case'*, *'has-two-vowels'*, *'contains-slash'*, and *'token-length'*, as well as binary features for the presence of certain words like *Chapter*, *on*, or *king* in the surrounding context. Sproat et al. (2001) also included a rough-draft rule-based classifier, which used hand-written regular expression to classify many of the number NSWs. The output of this rough-draft classifier was used as just another feature in the main classifier.

In order to build such a main classifier, we need a hand-labeled training set, in which each token has been labeled with its NSW category; one such hand-labeled data-base was produced by Sproat et al. (2001). Given such a labeled training set, we can use any supervised machine learning algorithm to build the classifier.

Formally, we can model this task as the goal of producing the tag sequence T which is most probable given the observation sequence:

$$T^* = \operatorname*{argmax}_T P(T|O) \tag{8.6}$$

One way to estimate this probability is via decision trees. For example, for each observed token o_i, and for each possible NSW tag t_j, the decision tree produces the posterior probability $P(t_j|o_i)$. If we make the incorrect but simplifying assumption that each tagging decision is independent of its neighbors, we can predict the best tag sequence $\hat{T} = argmax_T P(T|O)$ using the tree:

$$\hat{T} = \operatorname*{argmax}_T P(T|O)$$

$$\approx \prod_{i=1}^{m} \operatorname*{argmax}_{t} P(t|o_i) \tag{8.7}$$

The third step in dealing with NSWs is expansion into ordinary words. One NSW type, EXPN, is quite difficult to expand. These are the abbreviations and acronyms like **NY**. Generally these must be expanded by using an abbreviation dictionary, with any ambiguities dealt with by the homonym disambiguation algorithms discussed in the next section.

Expansion of the other NSW types is generally deterministic. Many expansions are trivial; for example, LSEQ expands to a sequence of words, one for each letter, ASWD expands to itself, NUM expands to a sequence of words representing the cardinal number, NORD expands to a sequence of words representing the ordinal number, and NDIG and NZIP both expand to a sequence of words, one for each digit.

Hundreds digits

Trailing unit digits

Other types are slightly more complex; NYER expands to two pairs of digits, unless the year ends in *00*, in which case the four years are pronounced as a cardinal number (*2000* as `two thousand`) or in the **hundreds** method (e.g., 1800 as `eighteen hundred`). NTEL can be expanded just as a sequence of digits; alternatively, the last four digits can be read as **paired digits**, in which each pair is read as an integer. It is also possible to read them in a form known as **trailing unit**, in which the digits are read serially until the last nonzero digit, which is pronounced followed by the appropriate unit (e.g., *876-5000* as `eight seven six five thousand`). The expansion of NDATE, MONEY, and NTIME is left as exercises (1)-(4) for the reader.

Of course many of these expansions are dialect-specific. In Australian English, the sequence *33* in a telephone number is generally read `double three`. Other languages also present additional difficulties in non-standard word normalization. In French or German, for example, in addition to the above issues, normalization may depend on morphological properties. In French, the phrase *1 fille* ('one girl') is normalized to `une fille`, but *1 garçon* ('one boy') is normalized to `un garçon`. Similarly, in German *Heinrich IV* ('Henry IV') can be normalized to `Heinrich der Vierte`, `Heinrich des Vierten`, `Heinrich dem Vierten`, or `Heinrich den Vierten` depending on the grammatical case of the noun (Demberg, 2006).

8.1.3 Homograph Disambiguation

The goal of our NSW algorithms in the previous section was to determine which sequence of standard words to pronounce for each NSW. But sometimes determining how to pronounce even standard words is difficult. This is particularly true for **homographs**, which are words with the same spelling but different pronunciations. Here are some examples of the English homographs *use*, *live*, and *bass*:

Homograph

(8.8) It's no use (/y uw s/) to ask to use (/y uw z/) the telephone.

(8.9) Do you live (/l ih v/) near a zoo with live (/l ay v/) animals?

(8.10) I prefer bass (/b ae s/) fishing to playing the bass (/b ey s/) guitar.

French homographs include *fils* (which has two pronunciations [fis] 'son' versus [fil] 'thread]), or the multiple pronunciations for *fier* ('proud' or 'to trust'), and *est* ('is' or 'East') (Divay and Vitale, 1997).

Luckily for the task of homograph disambiguation, the two forms of homographs in English (as well as in similar languages like French and German) tend to have different parts of speech.For example, the two forms of *use* above are (respectively) a noun and a verb, while the two forms of *live* are (respectively) a verb and a noun. Fig. 8.5 shows some interesting systematic relations between the pronunciation of some noun-verb and adj-verb homographs.

Final voicing			**Stress shift**			**-ate final vowel**		
	N (/s/)	**V** (/z/)		**N** (init. stress)	**V** (fin. stress)		**N/A** (final /ax/)	**V** (final /ey/)
use	y uw s	y uw z	record	r eh1 k axr0 d	r ix0 k ao1 r d	estimate	eh s t ih m ax t	eh s t ih m ey t
close	k l ow s	k l ow z	insult	ih1 n s ax0 l t	ix0 n s ah1 l t	separate	s eh p ax r ax t	s eh p ax r ey t
house	h aw s	h aw z	object	aa1 b j eh0 k t	ax0 b j eh1 k t	moderate	m aa d ax r ax t	m aa d ax r ey t

Figure 8.5 Some systematic relationships between homographs: final consonant (noun /s/ versus verb /z/), stress shift (noun initial versus verb final stress), and final vowel weakening in *-ate* noun/adjs.

Indeed, Liberman and Church (1992) showed that many of the most frequent homographs in 44 million words of AP newswire are disambiguatable just by using part-of-speech (the most frequent 15 homographs in order are: *use*, *increase*, *close*, *record*, *house*, *contract*, *lead*, *live*, *lives*, *protest*, *survey*, *project*, *separate*, *present*, *read*).

Thus because knowledge of part-of-speech is sufficient to disambiguate many homographs, in practice we perform homograph disambiguation by storing distinct pronunciations for these homographs labeled by part-of-speech, and then running a part-of-speech tagger to choose the pronunciation for a given homograph in context.

There are a number of homographs, however, where both pronunciations have the same part-of-speech. We saw two pronunciations for *bass* (fish versus instrument) above. Other examples of these include *lead* (because there are two noun pronunciations, /l iy d/ (a leash or restraint) and /l eh d/ (a metal)). We can also think of the task of disambiguating certain abbreviations (mentioned early as NSW disambiguation) as homograph disambiguation. For example, *Dr.* is ambiguous between `doctor` and `drive`, and *St.* between `Saint` or `street`. Finally, there are some words that differ in capitalizations like *polish*/*Polish*, which are homographs only in situations like sentence beginnings or all-capitalized text.

In practice, these latter classes of homographs that cannot be resolved using part-of-speech are often ignored in TTS systems. Alternatively, we can attempt to resolve them using the word sense disambiguation algorithms that we will introduce in Ch. 20, like the **decision-list** algorithm of Yarowsky (1997).

8.2 Phonetic Analysis

The next stage in synthesis is to take the normalized word strings from text analysis and produce a pronunciation for each word. The most important component here is a large pronunciation dictionary. Dictionaries alone turn out to be insufficient, because running text always contains words that don't appear in the dictionary. For example

Black et al. (1998) used a British English dictionary, the OALD lexicon on the first section of the Penn Wall Street Journal Treebank. Of the 39923 words (tokens) in this section, 1775 word tokens (4.6%) were not in the dictionary, of which 943 are unique (i.e. 943 types). The distributions of these unseen word tokens was as follows:

names	unknown	typos and other
1360	351	64
76.6%	19.8%	3.6%

Thus the two main areas where dictionaries need to be augmented is in dealing with names and with other unknown words. We'll discuss dictionaries in the next section, followed by names, and then turn to grapheme-to-phoneme rules for dealing with other unknown words.

8.2.1 Dictionary Lookup

Phonetic dictionaries were introduced in Sec. 7.5 of Ch. 8. One of the most widely-used for TTS is the freely available CMU Pronouncing Dictionary (CMU, 1993), which has pronunciations for about 120,000 words. The pronunciations are roughly phonemic, from a 39-phone ARPAbet-derived phoneme set. Phonemic transcriptions means that instead of marking surface reductions like the reduced vowels [ax] or [ix], CMUdict marks each vowel with a stress tag, 0 (unstressed), 1 (stressed), or 2 (secondary stress). Thus (non-diphthong) vowels with 0 stress generally correspond to [ax] or [ix]. Most words have only a single pronunciation, but about 8,000 of the words have two or even three pronunciations, and so some kinds of phonetic reductions are marked in these pronunciations. The dictionary is not syllabified, although the nucleus is implicitly marked by the (numbered) vowel. Fig. 8.2.1 shows some sample pronunciations.

ANTECEDENTS	AE2 N T IH0 S IY1 D AH0 N T S	*PAKISTANI*	P AE2 K IH0 S T AE1 N IY0
CHANG	CH AE1 NG	*TABLE*	T EY1 B AH0 L
DICTIONARY	D IH1 K SH AH0 N EH2 R IY0	*TROTSKY*	T R AA1 T S K IY2
DINNER	D IH1 N ER0	*WALTER*	W AO1 L T ER0
LUNCH	L AH1 N CH	*WALTZING*	W AO1 L T S IH0 NG
MCFARLAND	M AH0 K F AA1 R L AH0 N D	*WALTZING(2)*	W AO1 L S IH0 NG

Figure 8.6 Some sample pronunciations from the CMU Pronouncing Dictionary.

The CMU dictionary was designed for speech recognition rather than synthesis uses; thus it does not specify which of the multiple pronunciations to use for synthesis, does not mark syllable boundaries, and because it capitalizes the dictionary headwords, does not distinguish between e.g., *US* and *us* (the form *US* has the two pronunciations [AH1 S] and [Y UW1 EH1 S].

The 110,000 word UNISYN dictionary, freely available for research purposes, resolves many of these issues as it was designed specifically for synthesis (Fitt, 2002). UNISYN gives syllabifications, stress, and some morphological boundaries. Furthermore, pronunciations in UNISYN can also be read off in any of dozens of dialects of English, including General American, RP British, Australia, and so on. The UNISYN uses a slightly different phone set; here are some examples:

```
going:    { g * ou }.> i ng >
antecedents:   { * a n . tˆ i . s ˜ ii . d n! t }> s >
dictionary:    { d * i k . sh @ . n ˜ e . r ii }
```

8.2.2 Names

As the error analysis above indicated, names are an important issue in speech synthesis. The many types can be categorized into personal names (first names and surnames), geographical names (city, street, and other place names), and commercial names (company and product names). For personal names alone, Spiegel (2003) gives an estimate from Donnelly and other household lists of about two million different surnames and 100,000 first names just for the United States. Two million is a very large number; an order of magnitude more than the entire size of the CMU dictionary. For this reason, most large-scale TTS systems include a large name pronunciation dictionary. As we saw in Fig. 8.2.1 the CMU dictionary itself contains a wide variety of names; in particular it includes the pronunciations of the most frequent 50,000 surnames from an old Bell Lab estimate of US personal name frequency, as well as 6,000 first names.

How many names are sufficient? Liberman and Church (1992) found that a dictionary of 50,000 names covered 70% of the name tokens in 44 million words of AP newswire. Interestingly, many of the remaining names (up to 97.43% of the tokens in their corpus) could be accounted for by simple modifications of these 50,000 names. For example, some name pronunciations can be created by adding simple stress-neutral suffixes like `s` or `ville` to names in the 50,000, producing new names as follows:

```
walters = walter+s    lucasville = lucas+ville    abelson = abel+son
```

Other pronunciations might be created by rhyme analogy. If we have the pronunciation for the name *Trotsky*, but not the name *Plotsky*, we can replace the initial /tr/ from *Trotsky* with initial /pl/ to derive a pronunciation for *Plotsky*.

Techniques such as this, including morphological decomposition, analogical formation, and mapping unseen names to spelling variants already in the dictionary (Fackrell and Skut, 2004), have achieved some success in name pronunciation. In general, however, name pronunciation is still difficult. Many modern systems deal with unknown names via the grapheme-to-phoneme methods described in the next section, often by building two predictive systems, one for names and one for non-names. Spiegel (2003, 2002) summarizes many more issues in proper name pronunciation.

8.2.3 Grapheme-to-Phoneme

Grapheme-to-phoneme

Once we have expanded non-standard words and looked them all up in a pronunciation dictionary, we need to pronounce the remaining, unknown words. The process of converting a sequence of letters into a sequence of phones is called **grapheme-to-phoneme** conversion, sometimes shortened **g2p**. The job of a grapheme-to-phoneme algorithm is thus to convert a letter string like *cake* into a phone string like `[K EY K]`.

Letter-to-sound

The earliest algorithms for grapheme-to-phoneme conversion were rules written by hand using the Chomsky-Halle phonological rewrite rule format of Eq. 7.1 in Ch. 7. These are often called **letter-to-sound** or LTS rules, and they are still used in some systems. LTS rules are applied in order, with later (default) rules only applying if the context for earlier rules are not applicable. A simple pair of rules for pronouncing the letter *c* might be as follows:

(8.11) $c \rightarrow$ [k] / ___ {a,o}V ; context-dependent

(8.12) $c \rightarrow$ [s] ; context-independent

Actual rules must be much more complicated (for example *c* can also be pronounced [ch] in *cello* or *concerto*). Even more complex are rules for assigning stress, which are famously difficult for English. Consider just one of the many stress rules from Allen et al. (1987), where the symbol *X* represents all possible syllable onsets:

(8.13) $V \rightarrow$ [+stress] / X __C* $\{V_{\text{short}}\ C\ C?|V\}\ \{V_{\text{short}}\ C^*|V\}$

This rule represents the following two situations:

1. Assign 1-stress to the vowel in a syllable preceding a weak syllable followed by a morpheme-final syllable containing a short vowel and 0 or more consonants (e.g. *difficult*)
2. Assign 1-stress to the vowel in a syllable preceding a weak syllable followed by a morpheme-final vowel (e.g. *oregano*)

While some modern systems still use such complex hand-written rules, most systems achieve higher accuracy by relying instead on automatic or semi-automatic methods based on machine learning. This modern probabilistic grapheme-to-phoneme problem was first formalized by Lucassen and Mercer (1984). Given a letter sequence *L*, we are searching for the most probable phone sequence *P*:

(8.14) $$\hat{P} = \operatorname*{argmax}_P P(P|L)$$

The probabilistic method assumes a training set and a test set; both sets are lists of words from a dictionary, with a spelling and a pronunciation for each word. The next subsections show how the popular **decision tree** model for estimating this probability $P(P|L)$ can be trained and applied to produce the pronunciation for an unseen word.

Finding a letter-to-phone alignment for the training set

Most letter-to-phone algorithms assume that we have an **alignment**, which tells us which phones align with each letter. We'll need this alignment for each word in the training set. Some letters might align to multiple phones (e.g., *x* often aligns to k s), while other letters might align with no phones at all, like the final letter of *cake* in the following alignment:

L:	c	a	k	e
	|	|	|	|
P:	K	EY	K	ϵ

One method for finding such a letter-to-phone alignment is the semi-automatic method of (Black et al., 1998). Their algorithm is semi-automatic because it relies on a hand-written list of the **allowable** phones that can realize each letter. Here are allowables lists for the letters *c* and *e*:

c: `k ch s sh t-s` ϵ
e: `ih iy er ax ah eh ey uw ay ow y-uw oy aa` ϵ

In order to produce an alignment for each word in the training set, we take this allowables list for all the letters, and for each word in the training set, we find all alignments between the pronunciation and the spelling that conform to the allowables list. From this large list of alignments, we compute, by summing over all alignments for all words, the total count for each letter being aligned to each phone (or multi-phone or ϵ). From these counts we can normalize to get for each phone p_i and letter l_j a probability $P(p_i|l_j)$:

$$P(p_i|l_j) = \frac{\text{count}(p_i, l_j)}{\text{count}(l_j)} \tag{8.15}$$

We can now take these probabilities and realign the letters to the phones, using the Viterbi algorithm to produce the best (Viterbi) alignment for each word, where the probability of each alignment is just the product of all the individual phone/letter alignments.

In this way we can produce a single good alignment A for each particular pair (P,L) in our training set.

Choosing the best phone string for the test set

Given a new word *w*, we now need to map its letters into a phone string. To do this, we'll first train a machine learning classifier, like a decision tree, on the aligned training set. The job of the classifier will be to look at a letter of the word and generate the most probable phone.

What features should we use in this decision tree besides the aligned letter l_i itself? Obviously we can do a better job of predicting the phone if we look at a window of surrounding letters; for example consider the letter *a*. In the word *cat*, the *a* is pronounce AE. But in our word *cake*, *a* is pronounced EY, because *cake* has a final *e*; thus knowing whether there is a final *e* is a useful feature. Typically we look at the *k* previous letters and the *k* following letters.

Another useful feature would be the correct identity of the previous phone. Knowing this would allow us to get some phonotactic information into our probability model. Of course, we can't know the true identity of the previous phone, but we can approximate this by looking at the previous phone that was predicted by our model. In order to do this, we'll need to run our decision tree left to right, generating phones one by one.

In summary, in the most common decision tree model, the probability of each phone p_i is estimated from a window of *k* previous and *k* following letters, as well as the most recent *k* phones that were previously produced.

Fig. 8.7 shows a sketch of this left-to-right process, indicating the features that a decision tree would use to decide the letter corresponding to the letter *s* in the word *Jurafsky*. As this figure indicates, we can integrate stress prediction into phone prediction by augmenting our set of phones with stress information. We can do this by

having two copies of each vowel (e.g., AE and AE1), or possibly even the three levels of stress AE0, AE1, and AE2, that we saw in the CMU lexicon. We'll also want to add other features into the decision tree, including the part-of-speech tag of the word (most part-of-speech taggers provide an estimate of the part-of-speech tag even for unknown words) and facts such as whether the previous vowel was stressed.

In addition, grapheme-to-phoneme decision trees can also include other more sophisticated features. For example, we can use classes of letters (corresponding roughly to consonants, vowels, liquids, and so on). In addition, for some languages, we need to know features about the following word. For example French has a phenomenon called **liaison**, in which the realization of the final phone of some words depends on whether there is a next word, and whether it starts with a consonant or a vowel. For example the French word *six* can be pronounced [sis] (in *j'en veux six* 'I want six'), [siz] (*six enfants* 'six children'), [si] (*six filles* 'six girls').

Liaison

Finally, most synthesis systems build two separate grapheme-to-phoneme decision trees, one for unknown personal names and one for other unknown words. For pronouncing personal names it turns out to be helpful to use additional features that indicate which foreign language the names originally come from. Such features could be the output of a foreign-language classifier based on letter sequences (different languages have characteristic letter N-gram sequences).

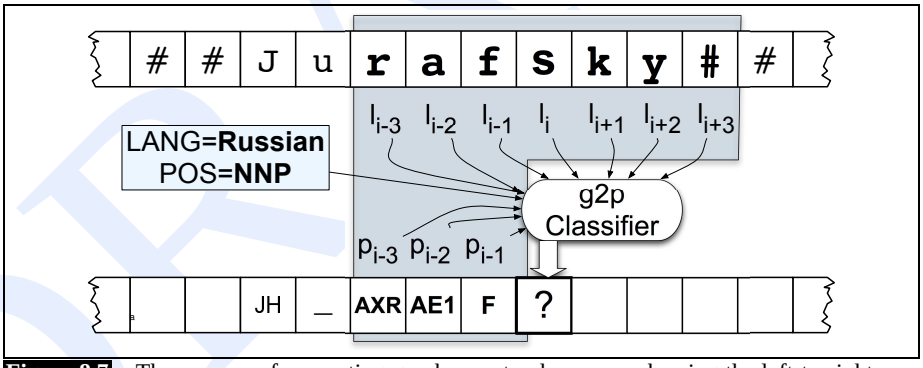


Figure 8.7 The process of converting graphemes to phonemes, showing the left-to-right process making a decision for the letter *s*. The features used by the decision tree are shown in blue. We have shown the context window $k = 3$; in real TTS systems the window size is likely to be 5 or even larger.

The decision tree is a conditional classifier, computing the phoneme string that has the highest conditional probability given the grapheme sequence. More recent grapheme-to-phoneme conversion makes use of a joint classifier, in which the hidden state is a combination of phone and grapheme called a **graphone**; see the end of the chapter for references.

8.3 Prosodic Analysis

Prosody

The final stage of linguistic analysis is prosodic analysis. In poetry, the word **prosody** refers to the study of the metrical structure of verse. In linguistics and language processing, however, we use the term **prosody** to mean the study of the intonational and rhythmic aspects of language. More technically, prosody has been defined by Ladd (1996) as the 'use of suprasegmental features to convey sentence-level pragmatic meanings'. The term **suprasegmental** means above and beyond the level of the segment or phone, and refers especially to the uses of acoustic features like **F0 duration**, and **energy** independently of the phone string.

Suprasegmental

By **sentence-level pragmatic meaning**, Ladd is referring to a number of kinds of meaning that have to do with the relation between a sentence and its discourse or external context. For example, prosody can be used to mark **discourse structure or function**, like the difference between statements and questions, or the way that a conversation is structured into segments or subdialogs. Prosody is also used to mark **saliency**, such as indicating that a particular word or phrase is important or salient. Finally, prosody is heavily used for affective and emotional meaning, such as expressing happiness, surprise, or anger.

In the next sections we will introduce the three aspects of prosody, each of which is important for speech synthesis: **prosodic prominence**, **prosodic structure** and **tune**. Prosodic analysis generally proceeds in two parts. First, we compute an abstract representation of the prosodic prominence, structure and tune of the text. For unit selection synthesis, this is all we need to do in the text analysis component. For diphone and HMM synthesis, we have one further step, which is to predict **duration** and **F0** values from these prosodic structures.

8.3.1 Prosodic Structure

Spoken sentences have prosodic structure in the sense that some words seem to group naturally together and some words seem to have a noticeable break or disjuncture between them. Often prosodic structure is described in terms of **prosodic phrasing**, meaning that an utterance has a prosodic phrase structure in a similar way to it having a syntactic phrase structure. For example, in the sentence *I wanted to go to London, but could only get tickets for France* there seems to be two main **intonation phrases**, their boundary occurring at the comma. Furthermore, in the first phrase, there seems to be another set of lesser prosodic phrase boundaries (often called **intermediate phrase**s) that split up the words as follows *I wanted* | *to go* | *to London*.

Prosodic Phrasing

Intonation phrase

intermediate phrase

Prosodic phrasing has many implications for speech synthesis; the final vowel of a phrase is longer than usual, we often insert a pause after an intonation phrases, and, as we will discuss in Sec. 8.3.6, there is often a slight drop in F0 from the beginning of an intonation phrase to its end, which resets at the beginning of a new intonation phrase.

Practical phrase boundary prediction is generally treated as a binary classification task, where we are given a word and we have to decide whether or not to put a prosodic boundary after it. A simple model for boundary prediction can be based on deterministic rules. A very high-precision rule is the one we saw for sentence segmentation: insert

a boundary after punctuation. Another commonly used rule inserts a phrase boundary before a function word following a content word.

More sophisticated models are based on machine learning classifiers. To create a training set for classifiers, we first choose a corpus, and then mark every prosodic boundaries in the corpus. One way to do this prosodic boundary labeling is to use an intonational model like ToBI or Tilt (see Sec. 8.3.4), have human labelers listen to speech and label the transcript with the boundary events defined by the theory. Because prosodic labeling is extremely time-consuming, however, a text-only alternative is often used. In this method, a human labeler looks only at the text of the training corpus, ignoring the speech. The labeler marks any juncture between words where they feel a prosodic boundary might legitimately occur if the utterance were spoken.

Given a labeled training corpus, we can train a decision tree or other classifier to make a binary (boundary vs. no boundary) decision at every juncture between words (Wang and Hirschberg, 1992; Ostendorf and Veilleux, 1994; Taylor and Black, 1998).

Features that are commonly used in classification include:

- **Length features**: phrases tend to be of roughly equal length, and so we can use various feature that hint at phrase length (Bachenko and Fitzpatrick, 1990; Grosjean et al., 1979; Gee and Grosjean, 1983).
  - The total number of words and syllables in utterance
  - The distance of the juncture from the beginning and end of the sentence (in words or syllables)
  - The distance in words from the last punctuation mark
- **Neighboring part-of-speech and punctuation**:
  - The part-of-speech tags for a window of words around the juncture. Generally the two words before and after the juncture are used.
  - The type of following punctuation

There is also a correlation between prosodic structure and the **syntactic structure** that will be introduced in Ch. 12, Ch. 13, and Ch. 14 (Price et al., 1991). Thus robust parsers like Collins (1997) can be used to label the sentence with rough syntactic information, from which we can extract syntactic features such as the size of the biggest syntactic phrase that ends with this word (Ostendorf and Veilleux, 1994; Koehn et al., 2000).

8.3.2 Prosodic prominence

Prominence

In any spoken utterance, some words sound more **prominent** than others. Prominent words are perceptually more salient to the listener; speakers make a word more salient in English by saying it louder, saying it slower (so it has a longer duration), or by varying F0 during the word, making it higher or more variable.

Pitch accent

We generally capture the core notion of prominence by associating a linguistic marker with prominent words, a marker called **pitch accent**. Words which are prominent are said to **bear** (be associated with) a pitch accent. Pitch accent is thus part of the phonological description of a word in context in a spoken utterance.

Pitch accent is related to **stress**, which we discussed in Ch. 7. The stressed syllable of a word is where pitch accent is realized. In other words, if a speaker decides to

highlight a word by giving it a pitch accent, the accent will appear on the stressed syllable of the word.

The following example shows accented words in capital letters, with the stressed syllable bearing the accent (the louder, longer, syllable) in boldface:

(8.16) I'm a little SUR**PRISED** to hear it **CHAR**ACTERIZED as UP**BEAT**.

Note that the function words tend not to bear pitch accent, while most of the content words are accented. This is a special case of the more general fact that very informative words (content words, and especially those that are new or unexpected) tend to bear accent (Ladd, 1996; Bolinger, 1972).

We've talked so far as if we only need to make a binary distinction between accented and unaccented words. In fact we generally need to make more fine-grained distinctions. For example the last accent in a phrase generally is perceived as being more prominent than the other accents. This prominent last accent is called the **nuclear accent**. Emphatic accents like nuclear accent are generally used for semantic purposes, for example to indicate that a word is the **semantic focus** of the sentence (see Ch. 21) or that a word is contrastive or otherwise important in some way. Such emphatic words are the kind that are often written IN CAPITAL LETTERS or with **STARS** around them in SMS or email or *Alice in Wonderland*; here's an example from the latter:

Nuclear accent

(8.17) 'I know SOMETHING interesting is sure to happen,' she said to herself,

Another way that accent can be more complex than just binary is that some words can be **less** prominent than usual. We introduced in Ch. 7 the idea that function words are often phonetically very **reduced**.

A final complication is that accents can differ according to the **tune** associated with them; for example accents with particularly high pitch have different functions than those with particularly low pitch; we'll see how this is modeled in the ToBI model in Sec. 8.3.4.

Ignoring tune for the moment, we can summarize by saying that speech synthesis systems can use as many as four levels of prominence: **emphatic accent**, **pitch accent**, **unaccented**, and **reduced**. In practice, however, many implemented systems make do with a subset of only two or three of these levels.

Let's see how a 2-level system would work. With two-levels, pitch accent prediction is a binary classification task, where we are given a word and we have to decide whether it is accented or not.

Since content words are very often accented, and function words are very rarely accented, the simplest accent prediction system is just to accent all content words and no function words. In most cases better models are necessary.

In principle accent prediction requires sophisticated semantic knowledge, for example to understand if a word is new or old in the discourse, whether it is being used contrastively, and how much new information a word contains. Early models made use of sophisticated linguistic models of all of this information (Hirschberg, 1993). But Hirschberg and others showed better prediction by using simple, robust features that correlate with these sophisticated semantics.

For example, the fact that new or unpredictable information tends to be accented can be modeled by using robust features like N-grams or TF*IDF (Pan and Hirschberg,

2000; Pan and McKeown, 1999). The unigram probability of a word $P(w_i)$ and its bigram probability $P(w_i|w_{i-1})$, both correlate with accent; the more probable a word, the less likely it is to be accented. Similarly, an information-retrieval measure known as

*TF*IDF*

TF*IDF (Term-Frequency/Inverse-Document Frequency; see Ch. 23) is a useful accent predictor. TF*IDF captures the semantic importance of a word in a particular document d, by downgrading words that tend to appear in lots of different documents in some large background corpus with N documents. There are various versions of TF*IDF; one version can be expressed formally as follows, assuming Nw is the frequency of w in the document d, and k is the total number of documents in the corpus that contain w:

$$\text{TF*IDF}(w) = Nw \times \log(\frac{N}{k}) \tag{8.18}$$

Accent ratio

For words which have been seen enough times in a training set, the **accent ratio** feature can be used, which models a word's individual probability of being accented. The accent ratio of a word is equal to the estimated probability of the word being accented if this probability is significantly different from 0.5, and equal to 0.5 otherwise. More formally,

$$AccentRatio(w) = \begin{cases} \frac{k}{N} & \texttt{if } B(k,N,0.5) \leq 0.05 \\ 0.5 & \texttt{otherwise} \end{cases}$$

where N is the total number of times the word w occurred in the training set, k is the number of times it was accented, and $B(k,n,0.5)$ is the probability (under a binomial distribution) that there are k successes in n trials if the probability of success and failure is equal (Nenkova et al., 2007; Yuan et al., 2005).

Features like part-of-speech, N-grams, TF*IDF, and accent ratio can then be combined in a decision tree to predict accents. While these robust features work relatively well, a number of problems in accent prediction still remain the subject of research.

For example, it is difficult to predict which of the two words should be accented in adjective-noun or noun-noun compounds. Some regularities do exist; for example adjective-noun combinations like *new truck* are likely to have accent on the right word (*new TRUCK*), while noun-noun compounds like *TREE surgeon* are likely to have accent on the left. But the many exceptions to these rules make accent prediction in noun compounds quite complex. For example the noun-noun compound *APPLE cake* has the accent on the first word while the noun-noun compound *apple PIE* or *city HALL* both have the accent on the second word (Liberman and Sproat, 1992; Sproat, 1994, 1998a).

Another complication has to do with rhythm; in general speakers avoid putting

Clash
Lapse

accents too close together (a phenomenon known as **clash**) or too far apart (**lapse**). Thus *city HALL* and *PARKING lot* combine as *CITY hall PARKING lot* with the accent on *HALL* shifting forward to *CITY* to avoid the clash with the accent on *PARKING* (Liberman and Prince, 1977),

Some of these rhythmic constraints can be modeled by using machine learning techniques that are more appropriate for sequence modeling. This can be done by running a decision tree classifier left to right through a sentence, and using the output of the previous word as a feature, or by using more sophisticated machine learning models like Conditional Random Fields (CRFs) (Gregory and Altun, 2004).

8.3.3 Tune

Tune

Two utterances with the same prominence and phrasing patterns can still differ prosodically by having different **tunes**. The **tune** of an utterance is the rise and fall of its F0 over time. A very obvious example of tune is the difference between statements and yes-no questions in English. The same sentence can be said with a final rise in F0 to indicate a yes-no-question, or a final fall in F0 to indicate a declarative intonation. Fig. 8.8 shows the F0 track of the same words spoken as a question or a statement. Note that the question rises at the end; this is often called a **question rise**. The falling intonation of the statement is called a **final fall**.

Question rise

Final fall

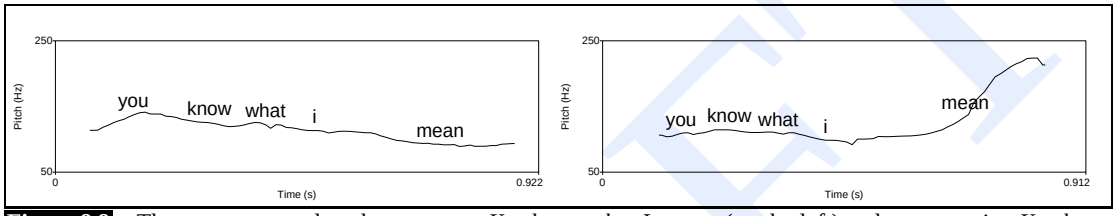


Figure 8.8 The same text read as the statement *You know what I mean.* (on the left) and as a question *You know what I mean?* (on the right). Notice that yes-no-question intonation in English has a sharp final rise in F0.

It turns out that English makes very wide use of tune to express meaning. Besides this well known rise for yes-no questions, an English phrase containing a list of nouns separated by commas often has a short rise called a **continuation rise** after each noun. English also has characteristic contours to express contradiction, to express surprise, and many more.

Continuation rise

The mapping between meaning and tune in English is extremely complex, and linguistic theories of intonation like ToBI have only begun to develop sophisticated models of this mapping. In practice, therefore, most synthesis systems just distinguish two or three tunes, such as the **continuation rise** (at commas), the **question rise** (at question mark if the question is a yes-no question), and a **final fall** otherwise.

8.3.4 More sophisticated models: ToBI

While current synthesis systems generally use simple models of prosody like the ones discussed above, recent research focuses on the development of much more sophisticated models. We'll very briefly discuss the **ToBI**, and **Tilt** models here.

ToBI

ToBI

One of the most widely used linguistic models of prosody is the **ToBI** (Tone and Break Indices) model (Silverman et al., 1992; Beckman and Hirschberg, 1994; Pierrehumbert, 1980; Pitrelli et al., 1994). ToBI is a phonological theory of intonation which models prominence, tune, and boundaries. ToBI's model of prominence and tunes is based on the 5 **pitch accents** and 4 **boundary tones** shown in Fig. 8.3.4.

An utterance in ToBI consists of a sequence of intonational phrases, each of which ends in one of the four **boundary tones**. The boundary tones are used to represent the

Boundary tone

Pitch Accents		Boundary Tones	
H*	peak accent	**L-L%**	"final fall": "declarative contour" of American English"
L*	low accent	**L-H%**	continuation rise
L*+H	scooped accent	**H-H%**	"question rise": cantonical yes-no question contour
L+H*	rising peak accent	**H-L%**	final level plateau (plateau because H- causes "upstep" of following)
H+!H*	step down		

Figure 8.9 The accent and boundary tones labels from the ToBI transcription system for American English intonation (Beckman and Ayers, 1997; Beckman and Hirschberg, 1994).

utterance final aspects of tune discussed in Sec. 8.3.3. Each word in the utterances can optionally be associated with one of the five types of pitch accents.

Each intonational phrase consists of one or more **intermediate phrase**. These phrases can also be marked with kinds of boundary tone, including the **%H** high initial boundary tone, which is used to mark a phrase which is particularly high in the speakers' pitch range, as well as final phrase accents **H-** and **L-**.

Break index

In addition to accents and boundary tones, ToBI distinguishes four levels of phrasing, which are labeled on a separate **break index** tier. The largest levels of phrasing are the intonational phrase (break index **4**) and the intermediate phrase (break index **3**), and were discussed above. Break index **2** is used to mark a disjuncture or pause between words that is smaller than an intermediate phrase, while **1** is used for normal phrase-medial word boundaries.

Tier

Fig. 8.10 shows the tone, orthographic, and phrasing **tiers** of a ToBI transcription, using the `praat` program. We see the same sentence read with two different intonation patterns. In (a), the word *Marianna* is spoken with a high H* accent, and the sentence has the declarative boundary tone L-L%. In (b), the word *Marianna* is spoken with a low L* accent and the yes-no question boundary tone H-H%. One goal of ToBI is to express different meanings to the different type of accents. Thus, for example, the L* accent adds a meaning of *surprise* to the sentence (i.e., with a connotation like 'Are you really saying it was Marianna?'). (Hirschberg and Pierrehumbert, 1986; Steedman, 2003).

ToBI models have been proposed for many languages, such as the J_TOBI system for Japanese (Venditti, 2005); see Jun (2005).

Other Intonation models

Tilt

The **Tilt** model (Taylor, 2000) resembles ToBI in using sequences of intonational events like accents and boundary tones. But Tilt does not use ToBI-style discrete phonemic classes for accents. Instead, each event is modeled by continuous parameters that represent the F0 shape of the accent.

Instead of giving each event a category label, as in ToBI, each Tilt prosodic event is characterized by a set of three acoustic parameters: the duration, the amplitude, and the **tilt** parameter. These acoustic parameters are trained on a corpus which has been hand-labeled for pitch accents (**a**) and boundary tones (**b**). The human labeling specifies the syllable which bears the accent or tone; the acoustic parameters are then trained

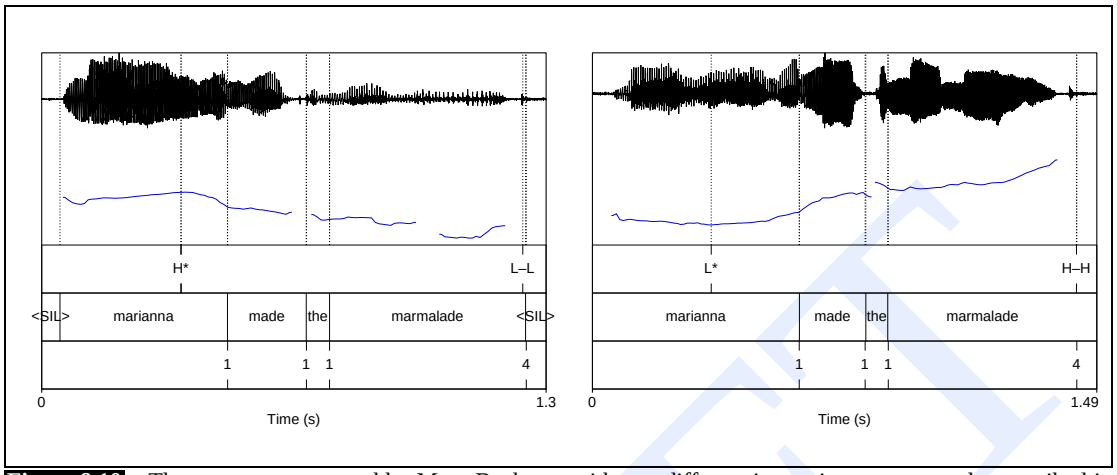


Figure 8.10 The same sentence read by Mary Beckman with two different intonation patterns and transcribed in ToBI. (a) shows an H* accent and the typical American English declarative final fall L-L%. (b) shows the L* accent, with the typical American English yes-no question rise H-H%.

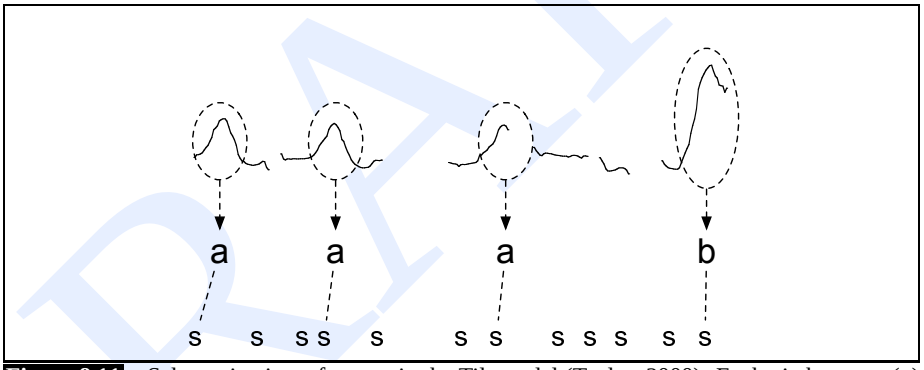


Figure 8.11 Schematic view of events in the Tilt model (Taylor, 2000). Each pitch accent (*a*) and boundary tone (*b*) is aligned with a syllable nucleus *s*.

automatically from the wavefile. Fig. 8.11 shows a sample of a Tilt representation. Each accent in Tilt is viewed as having a (possibly zero) **rise component** up to peak, followed by a (possible zero) **fall component**. An automatic accent detector finds the start, peak, and end point of each accent in the wavefile, which determines the duration and amplitude of the rise and fall components. The tilt parameter is an abstract description of the F0 slope of an event, calculated by comparing the relative sizes of the rise and fall for an event. A tilt value of 1.0 indicates a rise, tilt of -1.0 a fall, 0 equal rise and fall, -0.5 is an accent with a rise and a larger fall, and so on:

$$\begin{aligned} \text{tilt} &= \frac{\text{tilt}_{\text{amp}} + \text{tilt}_{\text{dur}}}{2} \\ &= \frac{|A_{\text{rise}}| - |A_{\text{fall}}|}{|A_{\text{rise}}| + |A_{\text{fall}}|} + \frac{D_{\text{rise}} - D_{\text{fall}}}{D_{\text{rise}} + D_{\text{fall}}} \end{aligned} \tag{8.19}$$

See the end of the chapter for pointers to other intonational models.

8.3.5 Computing duration from prosodic labels

The results of the text analysis processes described so far is a string of phonemes, annotated with words, with pitch accent marked on relevant words, and appropriate boundary tones marked. For the **unit selection** synthesis approaches that we will describe in Sec. 8.5, this is a sufficient output from the text analysis component.

For **diphone** synthesis, as well as other approaches like formant synthesis, we also need to specify the **duration** and the **F0** values of each segment.

Phones vary quite a bit in duration. Some of the duration is inherent to the identity of the phone itself. Vowels, for example, are generally much longer than consonants; in the Switchboard corpus of telephone speech, the phone [aa] averages 118 milliseconds, while [d] averages 68 milliseconds. But phone duration is also affected by a wide variety of contextual factors, which can be modeled by rule-based or statistical methods.

The most well-known of the rule-based methods is the method of Klatt (1979), which uses rules to model how the average or 'context-neutral' duration of a phone d is lengthened or shortened by context, while staying above a minimum duration d_{min}. Each Klatt rule is associated with a duration multiplicative factor; some examples:

Prepasual Lengthening:	The vowel or syllabic consonant in the syllable before a pause is lengthened by 1.4.
Non-phrase-final Shortening:	Segments which are not phrase-final are shortened by 0.6. Phrase-final postvocalic liquids and nasals are lengthened by 1.4.
Unstressed Shortening:	Unstressed segments are more compressible, so their minimum duration d_{min} is halved, and are shortened by .7 for most phone types.
Lengthening for Accent:	A vowel which bears accent is lengthened by 1.4
Shortening in Clusters:	A consonant followed by a consonant is shortened by 0.5.
Pre-voiceless shortening:	Vowels are shortened before a voiceless plosive by 0.7

Given the N factor weights f, the Klatt formula for the duration of a phone is:

$$d = d_{\text{min}} + \prod_{i=1}^{N} f_i \times (\bar{d} - d_{\text{min}}) \tag{8.20}$$

More recent machine-learning systems use the Klatt hand-written rules as the basis for defining features, for example using features such as the following:

- identity of the left and right context phone
- lexical stress and accent values of current phone
- position in syllable, word, phrase
- following pause

Sum-of-products

We can then train machine learning classifiers like decision trees or the **sum-of-products** model (van Santen, 1994, 1997, 1998), to combine the features to predict the final duration of the segment.

8.3.6 Computing F0 from prosodic labels

For diphone, articulatory, HMM, and formant synthesis we also need to specify the F0 values of each segment. For the tone sequence models like ToBI or Tilt, this F0 generation can be done by specifying F0 **target points** for each pitch accent and boundary tone; the F0 contour for the whole sentence can be created by interpolating among these targets (Anderson et al., 1984).

Target point

In order to specify a target point we need to describe what it is (the F0 value) and when it occurs (the exact time at which this peak or trough occurs in the syllable). The F0 values of the target points are generally not specified in absolute terms of Hertz. Instead, they are defined relative to **pitch range**. A speaker's **pitch range** is the range between the lowest frequency they use in a particular utterance (the **baseline frequency**) and the highest frequency in the utterance (the **topline**). In some models, target points are specified relative to a line in between called the **reference line**.

Pitch range

Baseline frequency

Topline

Reference line

For example, we might write a rule specifying that the very beginning of an utterance have a target point of 50% (halfway between the baseline and topline). In the rule-based system of Jilka et al. (1999) the target point for an H* accent is at 100% (the topline) and for an L* accent at 0% (at the baseline). L+H* accents have two target points, at 20% and 100%. Final boundary tones H-H% and L-L% are extra-high and extra-low at 120% and -20% respectively.

Second, we must also specify exactly where in the accented syllable the targets apply; this is known as accent **alignment**. In the rule-based system of Jilka et al. (1999), again, H* accents are aligned 60% of the way through the voiced part of the accent syllable (although IP-initial accents are aligned somewhat later in the syllable, while IP-final accents are aligned somewhat earlier).

Alignment

Instead of writing these rules by hand, the mapping from pitch accent sequence to F0 value may be learned automatically. For example Black and Hunt (1996) used linear regression to assign target values to each syllable. For each syllable with a pitch accent or boundary tone, they predicted three target values, at the beginning, middle, and end of the syllable. They trained three separate linear regression models, one for each of the three positions in the syllable. Features included:

- accent type on the current syllable as well as two previous and two following syllables
- lexical stress of this syllable and surrounding syllables
- number of syllables to start of phrase and to end of phrase
- number of accented syllables to end of phrase

Such machine learning models require a training set that is labeled for accent; a number of such prosodically-labeled corpora exist, although it is not clear how well these models generalize to unseen corpora.

Finally, F0 computation models must model the fact that pitch tends to decline through a sentence; this subtle drop in pitch across an utterance is called **declination**; an example is shown in Fig. 8.12.

Declination

The exact nature of declination is a subject of much research; in some models, it is treated by allowing the baseline (or both baseline and top-line) to decrease slowly over the utterance. In ToBI-like models, this downdrift in F0 is modeled by two sepa-

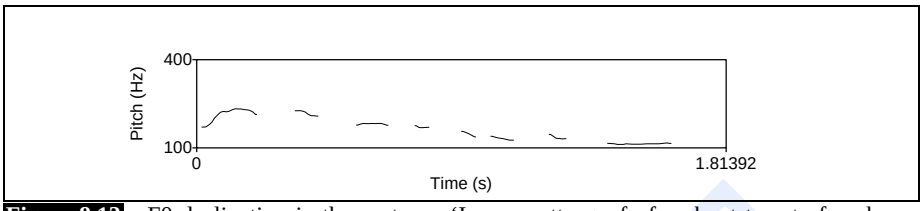


Figure 8.12 F0 declination in the sentence 'I was pretty goofy for about twenty-four hours afterwards'.

Downstep

rate components; in addition to declination, certain high tones are marked as carrying **downstep**. Each downstepped high accent causes the pitch range to be compressed, resulting in a lowered topline for each such accent.

8.3.7 Final result of text analysis: Internal Representation

The final output of text analysis is what we called the **internal representation** of the input text sentence. For unit selection synthesis, the internal representation can be as simple as a phone string together with indications of prosodic boundaries and prominent syllables, as shown in Fig. 8.1. For diphone synthesis as well as non-concatenative synthesis algorithms the internal representation must also include a duration and an F0 value for each phone.

Fig. 8.13 shows some sample TTS output from the FESTIVAL (Black et al., 1999) diphone speech synthesis system for the sentence *Do you really want to see all of it?*. This output, together with the F0 values shown in Fig. 8.14 would be the input to the **waveform synthesis** component described in Sec. 8.4. The durations here are computed by a CART-style decision tree (Riley, 1992).

				H*												L*		L- H%			
do		you		really				want				to		see		all		of		it	
d	uw	y	uw	r	ih	l	iy	w	aa	n	t	t	ax	s	iy	ao	l	ah	v	ih	t
110	110	50	50	75	64	57	82	57	50	72	41	43	47	54	130	76	90	44	62	46	220

Figure 8.13 Output of the FESTIVAL (Black et al., 1999) generator for the sentence *Do you really want to see all of it?*, together with the F0 contour shown in Fig. 8.14. Figure thanks to Paul Taylor.

As was suggested above, determining the proper prosodic pattern for a sentence is difficult, as real-world knowledge and semantic information is needed to know which syllables to accent, and which tune to apply. This sort of information is difficult to extract from the text and hence prosody modules often aim to produce a "neutral declarative" version of the input text, which assume the sentence should be spoken in a default way with no reference to discourse history or real-world events. This is one of the main reasons why intonation in TTS often sounds "wooden".

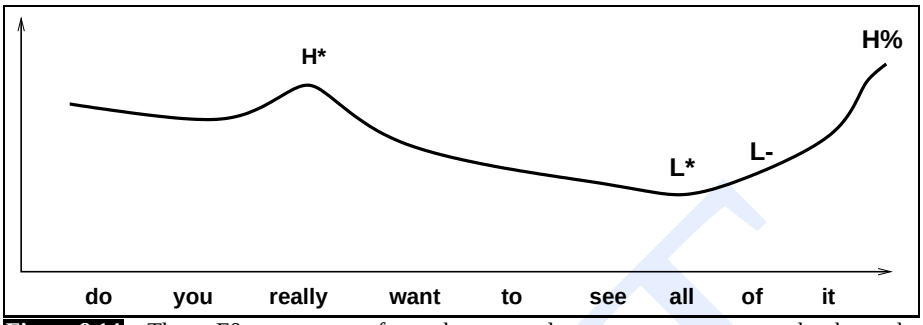


Figure 8.14 The F0 contour for the sample sentence generated by the FESTIVAL synthesis system in Fig. 8.13, thanks to Paul Taylor.

8.4 Diphone Waveform synthesis

We are now ready to see how the internal representation can be turned into a waveform. We will present two kinds of **concatentative** synthesis: **diphone synthesis** in this section, and **unit selection synthesis** in the next section.

Recall that for diphone synthesis, our internal representation is as shown in Fig. 8.13 and Fig. 8.14, consisting of a list of phones, each phone associated with a duration and a set of F0 targets.

Diphone

The diphone concatenative synthesis model generates a waveform from a sequence of phones by selecting and concatenating units from a prerecorded database of **diphones**. A diphone is a phone-like unit going from roughly the middle of one phone to the middle of the following phone. Diphone concatenative synthesis can be characterized by the following steps:

Training:

1. Record a single speaker saying an example of each diphone.
2. Cut each diphone out from the speech and store all diphones in a diphone database.

Synthesis:

1. Take from the database a sequence of diphones that corresponds to the desired phone sequence.
2. Concatenate the diphones, doing some slight signal processing at the boundaries
3. Use signal processing to change the prosody (f0, duration) of the diphone sequence to the desired prosody.

Coarticulation

We tend to use diphones rather than phones for concatenative synthesis because of the phenomenon of **coarticulation**. In Ch. 7 we defined **coarticulation** as the movement of articulators to anticipate the next sound, or perseverating movement from the last sound. Because of coarticulation, each phone differs slightly depending on the previous and following phone. This if we just concatenated phones together, we would have very large discontinuities at the boundaries.

In a diphone, we model this coarticulation by including the transition to the next phone inside the unit. The diphone [w-eh], for example, includes the transition from the [w] phone to the [eh] phone. Because a diphone is defined from the middle of one phone to the middle of the next, when we concatenate the diphones, we are concatenating the middle of phones, and the middle of phones tend to be less influenced by the context. Fig. 10.11 shows the intuition that the beginning and end of the vowel [eh] have much more movement than the center.

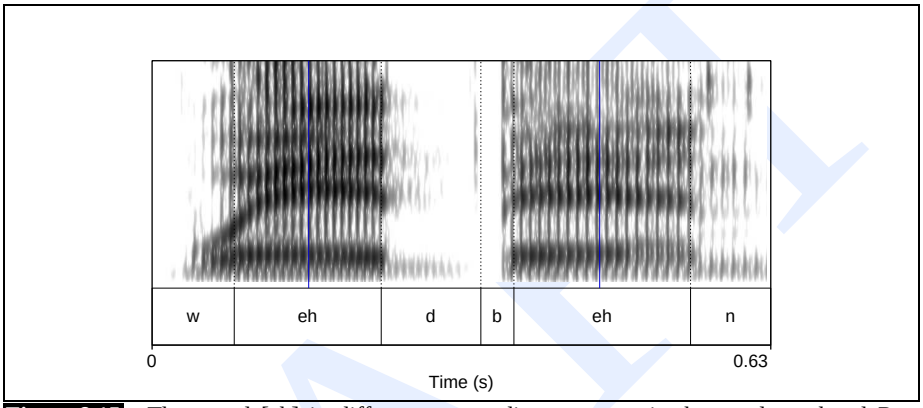


Figure 8.15 The vowel [eh] in different surrounding contexts, in the words *wed* and *Ben*. Notice the differences in the second formants (F2) at the beginning and end of the [eh], but the relatively steady state portion in the middle at the blue line.

8.4.1 Building a diphone database

There are six steps in building a diphone database:

1. Create a **diphone inventory**
2. Recruit a speaker
3. Create a text for the speaker to read for each diphone
4. Record the speaker reading each diphone
5. Segment, label, and pitch-mark the diphones
6. Excise the diphones

What is the inventory of diphones that we need for a system? If we have 43 phones (like the AT&T system of Olive et al. (1998)), there are $43^2 = 1849$ hypothetically possible diphone combinations. Not all of these diphones can actually occur. For example, English **phonotactic** constraints rule out some combinations; phones like [h], [y], and [w] can only occur before vowels. In addition, some diphone systems don't bother storing diphones if there is no possible coarticulation between the phones, such as across the silence between successive voiceless stops. The 43-phone system of Olive et al. (1998) thus has only 1162 diphones rather than the 1849 hypothetically possible set.

Voice talent

Next we recruit our speaker, often called a **voice talent**. The database of diphones

Voice

for this speaker is called a **voice**; commercial systems often have multiple voices, such as one male and one female voice.

We'll now create a text for the voice talent to say, and record each diphone. The most important thing in recording diphones is to keep them as consistent as possible; if possible, they should have constant pitch, energy, and duration, so they are easy to paste together without noticeable breaks. We do this by enclosing each diphone to be recorded in a **carrier phrase**. By putting the diphone in the middle of other phones, we keep utterance-final lengthening or initial phone effects from making any diphone louder or quieter than the others. We'll need different carrier phrases for consonant-vowel, vowel-consonant, phone-silence, and silence-phone sequences. For example, a consonant vowel sequence like [b aa] or [b ae] could be embedded between the syllables [t aa] and [m aa]:

Carrier phrase

pause t aa b aa m aa pause
pause t aa b ae m aa pause
pause t aa b eh m aa pause
...

If we have an earlier synthesizer voice lying around, we usually use that voice to read the prompts out loud, and have our voice talent repeat after the prompts. This is another way to keep the pronunciation of each diphone consistent. It is also very important to use a high quality microphone and a quiet room or, better, a studio sound booth.

Once we have recorded the speech, we need to label and segment the two phones that make up each diphone. This is usually done by running a speech recognizer in **forced alignment mode**. In forced alignment mode, a speech recognition is told exactly what the phone sequence is; its job is just to find the exact phone boundaries in the waveform. Speech recognizers are not completely accurate at finding phone boundaries, and so usually the automatic phone segmentation is hand-corrected.

We now have the two phones (for example [b aa]) with hand-corrected boundaries. There are two ways we can create the /b-aa/ diphone for the database. One method is to use rules to decide how far into the phone to place the diphone boundary. For example, for stops, we put place the diphone boundary 30% of the way into the phone. For most other phones, we place the diphone boundary 50% into the phone.

A more sophisticated way to find diphone boundaries is to store the entire two phones, and wait to excise the diphones until we are know what phone we are about to concatenate with. In this method, known as **optimal coupling**, we take the two (complete, uncut) diphones we need to concatenate, and we check every possible cutting point for each diphones, choosing the two cutting points that would make the final frame of the first diphone acoustically most similar to the end frame of the next diphone (Taylor and Isard, 1991; Conkie and Isard, 1996). Acoustical similar can be measured by using **cepstral similarity**, to be defined in Sec. 9.3.

Optimal coupling

8.4.2 Diphone concatenation and TD-PSOLA for prosody

We are now ready to see the remaining steps for synthesizing an individual utterance. Assume that we have completed text analysis for the utterance, and hence arrived at a

sequence of diphones and prosodic targets, and that we have also grabbed the appropriate sequence of diphones from the diphone database. Next we need to concatenate the diphones together and then adjust the prosody (pitch, energy, and duration) of the diphone sequence to match the prosodic requirements from the intermediate representation.

Given two diphones, what do we need to do to concatenate them successfully? If the waveforms of the two diphones edges across the juncture are very different, a perceptible **click** will result. Thus we need to apply a windowing function to the edge of both diphones so that the samples at the juncture have low or zero amplitude. Furthermore, if both diphones are voiced, we need to insure that the two diphones are joined **pitch-synchronously**. This means that the pitch periods at the end of the first diphone must line up with the pitch periods at the beginning of the second diphone; otherwise the resulting single irregular pitch period at the juncture is perceptible as well.

Click

Pitch-synchronous

Now given our sequence of concatenated diphones, how do we modify the pitch and duration to meet our prosodic requirements? It turns out there is a very simple algorithm for doing this called **TD-PSOLA** (**Time-Domain Pitch-Synchronous OverLap-and-Add**).

TD-PSOLA

As we just said, a **pitch-synchronous** algorithm is one in which we do something at each pitch period or **epoch**. For such algorithms it is important to have very accurate pitch markings: measurements of exactly where each pitch pulse or **epoch** occurs. An epoch can be defined by the instant of maximum glottal pressure, or alternatively by the instant of glottal closure. Note the distinction between **pitch marking** or **epoch detection** and **pitch tracking**. Pitch tracking gives the value of F0 (the average cycles per second of the glottis) at each particular point in time, averaged over a neighborhood. Pitch marking finds the exact point in time at each vibratory cycle at which the vocal folds reach some specific point (epoch).

Pitch marking

Pitch tracking

Epoch-labeling can be done in two ways. The traditional way, and still the most accurate, is to use an **electroglottograph** or **EGG** (often also called a **laryngograph** or **Lx**). An EGG is a device which straps onto the (outside of the) speaker's neck near the larynx and sends a small current through the Adam's apple. A transducer detects whether the glottis is open or closed by measuring the impedance across the vocal folds. Some modern synthesis databases are still recorded with an EGG. The problem with using an EGG is that it must be attached to the speaker while they are recording the database. Although an EGG isn't particularly invasive, this is still annoying, and the EGG must be used during recording; it can't be used to pitch-mark speech that has already been collected. Modern epoch detectors are now approaching a level of accuracy that EGGs are no longer used in most commercial TTS engines. Algorithms for epoch detection include Brookes and Loke (1999), Veldhuis (2000).

Electroglotto-graph

EGG

Laryngograph

Lx

Given an epoch-labeled corpus, the intuition of TD-PSOLA is that we can modify the pitch and duration of a waveform by extracting a frame for each pitch period (windowed so that the frame doesn't have sharp edges) and then recombining these frames in various ways by simply overlapping and adding the windowed pitch period frames (we will introduce the idea of windows in Sec. 9.3.2). The idea that we modify a signal by extracting frames, manipulating them in some way and then recombining them by adding up the overlapped signals is called the **overlap-and-add** or **OLA**

Overlap-and-add

OLA

algorithm; TD-PSOLA is a special case of overlap-and-add in which the frames are pitch-synchronous, and the whole process takes place in the time domain.

For example, in order to assign a specific duration to a diphone, we might want to lengthen the recorded master diphone. To lengthen a signal with TD-PSOLA, we simply insert extra copies of some of the pitch-synchronous frames, essentially duplicating a piece of the signal. Fig. 8.16 shows the intuition.

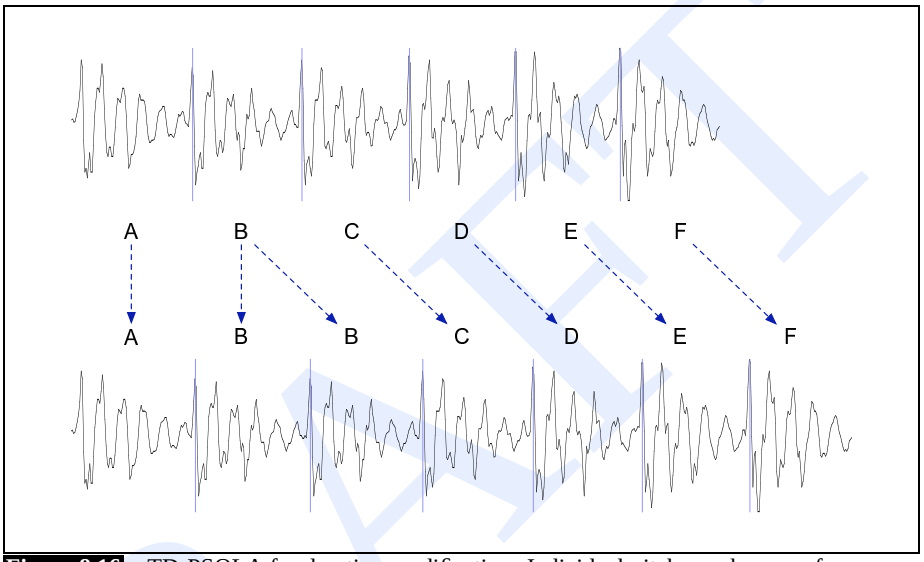


Figure 8.16 TD-PSOLA for duration modification. Individual pitch-synchronous frames can be duplicated to lengthen the signal (as shown here), or deleted to shorten the signal.

TD-PSOLA can also be used to change the F0 value of a recorded diphone to give a higher or lower value. To increase the F0, we extract each pitch-synchronous frame from the original recorded diphone signal, place the frames closer together (overlapping them), with the amount of overlap determined by the desired period and hence frequency, and then add up the overlapping signals to produce the final signal. But note that by moving all the frames closer together, we make the signal shorter in time! Thus in order to change the pitch while holding the duration constant, we need to add duplicate frames.

Fig. 8.17 shows the intuition; in this figure we have explicitly shown the extracted pitch-synchronous frames which are overlapped and added; note that the frames moved closer together (increasing the pitch) while extra frames have been added to hold the duration constant.

8.5 Unit Selection (Waveform) Synthesis

Diphone waveform synthesis suffers from two main problems. First, the stored diphone database must be modified by signal process methods like PSOLA to produce

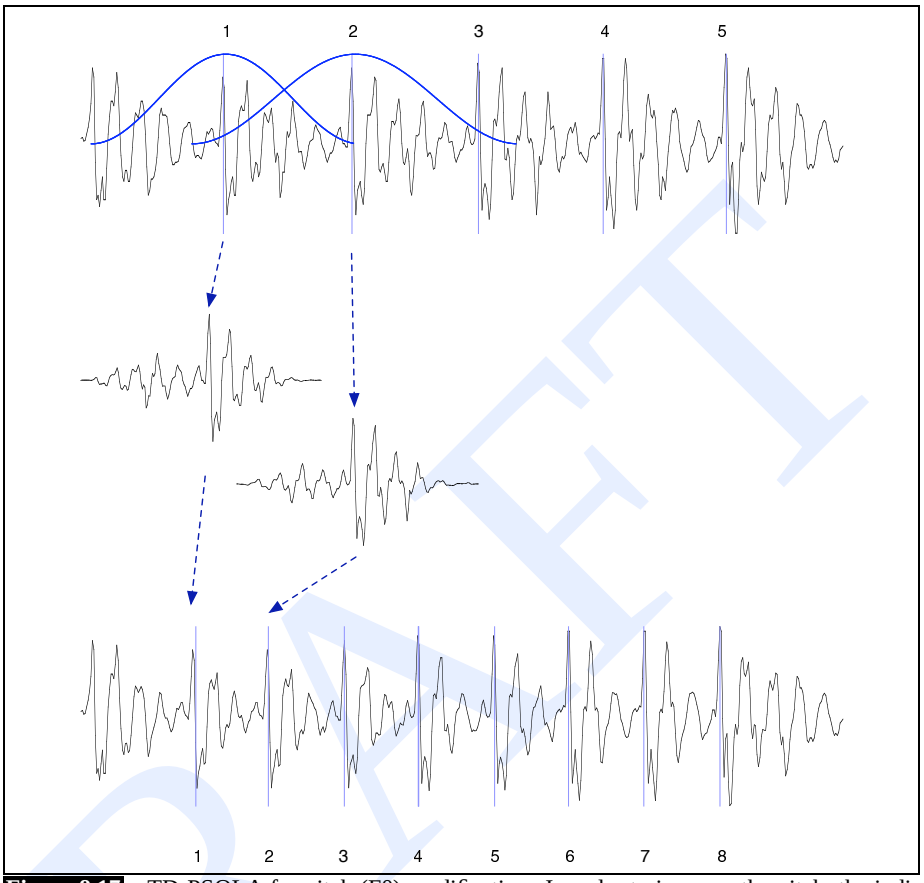


Figure 8.17 TD-PSOLA for pitch (F0) modification. In order to increase the pitch, the individual pitch-synchronous frames are extracted, Hanning windowed, moved closer together and then added up. To decrease the pitch, we move the frames further apart. Increasing the pitch will result in a shorter signal (since the frames are closer together), so we also need to duplicate frames if we want to change the pitch while holding the duration constant.

the desired prosody. Any kind of signal processing of the stored speech leaves artifacts in the speech which can make the speech sound unnatural. Second, diphone synthesis only captures the coarticulation due to a single neighboring phone. But there are many more global effects on phonetic realization, including more distant phones, syllable structure, the stress patterns of nearby phones, and even word-level effects.

Unit selection synthesis

For this reason, modern commercial synthesizers are based on a generalization of diphone synthesis called **unit selection synthesis**. Like diphone synthesis, unit selection synthesis is a kind of concatenative synthesis algorithm. It differs from classic diphone synthesis in two ways:

1. In diphone synthesis the database stores exactly one copy of each diphone, while in unit selection, the unit database is many hours long, containing many copies of each diphone.

2. In diphone synthesis, the prosody of the concatenated units is modified by PSOLA or similar algorithms, while in unit selection no (or minimal) signal processing is applied to the concatenated units.

The strengths of unit selection are due to the large unit database. In a sufficiently large database, entire words or phrases of the utterance we want to synthesize may be already present in the database, resulting in an extremely natural waveform for these words or phrases. In addition, in cases where we can't find a large chunk and have to back off to individual diphones, the fact that there are so many copies of each diphone makes it more likely that we will find one that will fit in very naturally.

The architecture of unit selection can be summarized as follows. We are given a large database of units; let's assume these are diphones (although it's also possible to do unit selection with other kinds of units such half-phones, syllables, or half-syllables). We are also given a characterization of the target 'internal representation', i.e. a phone string together with features such as stress values, word identity, F0 information, as described in Fig. 8.1.

The goal of the synthesizer is to select from the database the best sequence of diphone units that corresponds to the target representation. What do we mean by the 'best' sequence? Intuitively, the best sequence would be one in which:

- each diphone unit we select exactly meets the specifications of the target diphone (in terms of F0, stress level, phonetic neighbors, etc)
- each diphone unit concatenates smoothly with its neighboring units, with no perceptible break.

Of course, in practice, we can't guarantee that there wil be a unit which exactly meets our specifications, and we are unlikely to find a sequence of units in which every single join is imperceptible. Thus in practice unit selection algorithms implement a gradient version of these constraints, and attempt to find the sequence of unit which at least minimizes the **target cost** and the **join cost**:

Target cost

Join cost

Target cost $T(u_t, s_t)$:	how well the target specification s_t matches the potential unit u_t
Join cost $J(u_t, u_{t+1})$:	how well (perceptually) the potential unit u_t joins with its potential neighbor u_{t+1}

The T and J values are expressed as **costs** meaning that high values indicate bad matches and bad joins (Hunt and Black, 1996a).

Formally, then, the task of unit selection synthesis, given a sequence S of T target specifications, is to find the sequence $\hat{U}$ of T units from the database which minimizes the sum of these costs:

$$\hat{U} = \operatorname*{argmin}_{U} \sum_{t=1}^{T} T(s_t, u_t) + \sum_{t=1}^{T-1} J(u_t, u_{t+1}) \tag{8.21}$$

Let's now define the target cost and the join cost in more detail before we turn to the decoding and training tasks.

The target cost measures how well the unit matches the target diphone specification. We can think of the specification for each diphone target as a feature vector; here

are three sample vectors for three target diphone specifications, using dimensions (features) like *should the syllable be stressed*, and *where in the intonational phrase should the diphone come from*:

```
/ih-t/, +stress, phrase internal, high F0, content word
/n-t/, -stress, phrase final, high F0, function word
/dh-ax/, -stress, phrase initial, low F0, word 'the'
```

We'd like the distance between the target specification s and the unit to be some function of the how different the unit is on each of these dimensions from the specification. Let's assume that for each dimension p, we can come up with some **subcost** $T_p(s_t[p], u_j[p])$. The subcost for a binary feature like *stress* might be 1 or 0. The subcost for a continuous feature like F0 might be the difference (or log difference) between the specification F0 and unit F0. Since some dimensions are more important to speech perceptions than others, we'll also want to weight each dimension. The simplest way to combine all these subcosts is just to assume that they are independent and additive. Using this model, the total target cost for a given target/unit pair is the weighted sum over all these subcosts for each feature/dimension:

$$T(s_t, u_j) = \sum_{p=1}^{P} w_p T_p(s_t[p], u_j[p]) \tag{8.22}$$

The target cost is a function of the desired diphone specification and a unit from the database. The **join cost**, by contrast, is a function of two units from the database. The goal of the join cost is to be low (0) when the join is completely natural, and high when the join would be perceptible or jarring. We do this by measuring the acoustic similarity of the edges of the two units that we will be joining. If the two units have very similar energy, F0, and spectral features, they will probably join well. Thus as with the target cost, we compute a join cost by summing weighted subcosts:

$$J(u_t, u_{t+1}) = \sum_{p=1}^{P} w_p J_p(u_t[p], u_{t+1}[p]) \tag{8.23}$$

The three subcosts used in the classic Hunt and Black (1996b) algorithm are the **cepstral distance** at the point of concatenation, and the absolute differences in log power and F0. We will introduce the cepstrum in Sec. 9.3.

In addition, if the two units u_t and u_{t+1} to be concatenated were consecutive diphones in the unit database (i.e. they followed each other in the original utterance), then we set the join cost to 0: $J(u_t, u_{t+1}) = 0$. This is an important feature of unit selection synthesis, since it encourages large natural sequences of units to be selected from the database.

How do we find the best sequence of units which minimizes the sum of the target and join costs as expressed in Eq. 8.21? The standard method is to think of the unit selection problem as a Hidden Markov Model. The target units are the observed outputs, and the units in the database are the hidden states. Our job is to find the best hidden state sequence. We will use the Viterbi algorithm to solve this problem, just as we saw

it in Ch. 5 and Ch. 6, and will see it again in Ch. 9. Fig. 8.18 shows a sketch of the search space as well as the best (Viterbi) path that determines the best unit sequence.

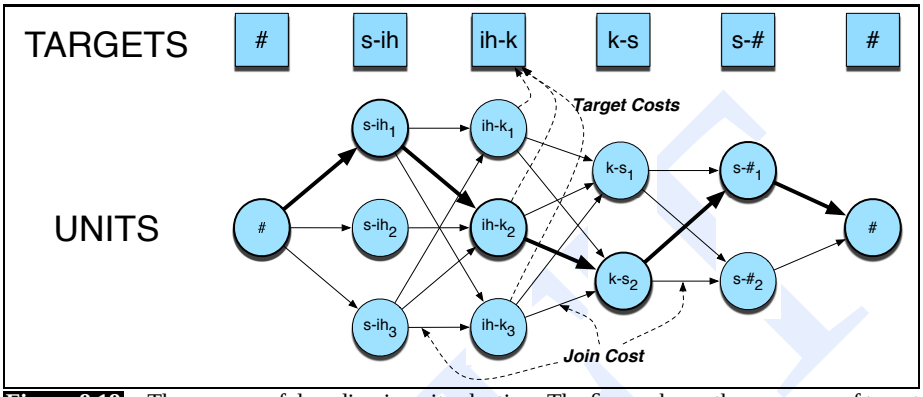


Figure 8.18 The process of decoding in unit selection. The figure shows the sequence of target (specification) diphones for the word *six*, and the set of possible database diphone units that we must search through. The best (Viterbi) path that minimizes the sum of the target and join costs is shown in bold.

The weights for join and target costs are often set by hand, since the number of weights is small (on the order of 20) and machine learning algorithms don't always achieve human performance. The system designer listens to entire sentences produced by the system, and chooses values for weights that result in reasonable sounding utterances. Various automatic weight-setting algorithms do exist, however. Many of these assume we have some sort of distance function between the acoustics of two sentences, perhaps based on cepstral distance. The method of Hunt and Black (1996b), for example, holds out a test set of sentences from the unit selection database. For each of these test sentences, we take the word sequence and synthesize a sentence waveform (using units from the other sentences in the training database). Now we compare the acoustics of the synthesized sentence with the acoustics of the true human sentence. Now we have a sequence of synthesized sentences, each one associated with a distance function to its human counterpart. Now we use linear regression based on these distances to set the target cost weights so as to minimize the distance.

There are also more advanced methods of assigning both target and join costs. For example, above we computed target costs between two units by looking at the features of the two units, doing a weighted sum of feature costs, and choosing the lowest-cost unit. An alternative approach (which the new reader might need to come back to after learning the speech recognition techniques introduced in the next chapters) is to map the target unit into some acoustic space, and then find a unit which is near the target in that acoustic space. In the method of Donovan and Eide (1998), Donovan and Woodland (1995), for example, all the training units are clustered using the decision tree algorithm of speech recognition described in Sec. 10.3. The decision tree is based on the same features described above, but here for each set of features, we follow a path down the decision tree to a leaf node which contains a cluster of units that have those features. This cluster of units can be parameterized by a Gaussian model, just as for speech recognition, so that we can map a set of features into a probability distribution

over cepstral values, and hence easily compute a distance between the target and a unit in the database. As for join costs, more sophisticated metrics make use of how perceivable a particular join might be (Wouters and Macon, 1998; Syrdal and Conkie, 2004; Bulyko and Ostendorf, 2001).

8.6 Evaluation

Speech synthesis systems are evaluated by human listeners. The development of a good automatic metric for synthesis evaluation, that would eliminate the need for expensive and time-consuming human listening experiments, remains an open and exiting research topic.

Intelligibility

The minimal evaluation metric for speech synthesis systems is **intelligibility**: the ability of a human listener to correctly interpret the words and meaning of the synthesized utterance. A further metric is **quality**; an abstract measure of the naturalness, fluency, or clarity of the speech.

Quality

Diagnostic Rhyme Test
DRT

The most local measures of intelligibility test the ability of a listener to discriminate between two phones. The **Diagnostic Rhyme Test** (**DRT**) (Voiers et al., 1975) tests the intelligibility of initial consonants. It is based on 96 pairs of confusable rhyming words which differ only in a single phonetic feature, such as (*dense/tense*) or *bond/pond* (differing in voicing) or *mean/beat* or *neck/deck* (differing in nasality), and so on. For each pair, listeners hear one member of the pair, and indicate which they think it is. The percentage of right answers is then used as an intelligibility metric. The **Modified Rhyme Test** (**MRT**) (House et al., 1965) is a similar test based on a different set of 300 words, consisting of 50 sets of 6 words. Each 6-word set differs in either initial or final consonants (e.g., *went*, *sent*, *bent*, *dent*, *tent*, *rent* or *bat*, *bad*, *back*, *bass*, *ban*, *bath*). Listeners are again given a single word and must identify from a closed list of six words; the percentage of correct identifications is again used as an intelligibility metric.

Modified Rhyme Test
MRT

Carrier phrase

Since context effects are very important, both DRT and MRT words are embedded in **carrier phrases** like the following:

```
Now we will say <word> again.
```

SUS

In order to test larger units than single phones, we can use **semantically unpredictable sentences** (**SUS**) (Benoît et al., 1996). These are sentences constructed by taking a simple POS template like DET ADJ NOUN VERB DET NOUN and inserting random English words in the slots, to produce sentences like

```
The unsure steaks closed the fish.
```

Measures of intelligibility like DRT/MRT and SUS are designed to factor out the role of context in measuring intelligibility. While this allows us to get a carefully controlled measure of a system's intelligibility, such acontextual or semantically unpredictable sentences aren't a good fit to how TTS is used in most commercial applications. Thus in commercial applications instead of DRT or SUS, we generally test intelligibility using situations that mimic the desired applications; reading addresses out loud, reading lines of news text, and so on.

MOS

To further evaluate the **quality** of the synthesized utterances, we can play a sentence for a listener and ask them to give a **mean opinion score** (**MOS**), a rating of how good the synthesized utterances are, usually on a scale from 1-5. We can then compare systems by comparing their MOS scores on the same sentences (using, e.g., t-tests to test for significant differences).

AB tests

If we are comparing exactly two systems (perhaps to see if a particular change actually improved the system), we can use **AB tests** In AB tests, we play the same sentence synthesized by two different systems (an A and a B system). The human listener chooses which of the two utterances they like better. We can do this for 50 sentences and compare the number of sentences preferred for each systems. In order to avoid ordering preferences, for each sentence we must present the two synthesized waveforms in random order.

Bibliographical and Historical Notes

As we noted at the beginning of the chapter, speech synthesis is one of the earliest fields of speech and language processing. The 18th century saw a number of physical models of the articulation process, including the von Kempelen model mentioned above, as well as the 1773 vowel model of Kratzenstein in Copenhagen using organ pipes.

But the modern era of speech synthesis can clearly be said to have arrived by the early 1950's, when all three of the major paradigms of waveform synthesis had been proposed (formant synthesis, articulatory synthesis, and concatenative synthesis).

Concatenative synthesis seems to have been first proposed by Harris (1953) at Bell Laboratories, who literally spliced together pieces of magnetic tape corresponding to phones. Harris's proposal was actually more like unit selection synthesis than diphone synthesis, in that he proposed storing multiple copies of each phone, and proposed the use of a join cost (choosing the unit with the smoothest formant transitions with the neighboring unit). Harris's model was based on the phone, rather than diphone, resulting in problems due to coarticulation. Peterson et al. (1958) added many of the basic ideas of unit selection synthesis, including the use of diphones, a database with multiple copies of each diphone with differing prosody, and each unit labeled with intonational features including F0, stress, and duration, and the use of join costs based on F0 and formant distant between neighboring units. They also proposed microconcatenation techniques like windowing the waveforms. The Peterson et al. (1958) model was purely theoretical, however, and concatenative synthesis was not implemented until the 1960's and 1970's, when diphone synthesis was first implemented (Dixon and Maxey, 1968; Olive, 1977). Later diphone systems included larger units such as consonant clusters (Olive and Liberman, 1979). Modern unit selection, including the idea of large units of non-uniform length, and the use of a target cost, was invented by Sagisaka (1988), Sagisaka et al. (1992). Hunt and Black (1996b) formalized the model, and put it in the form in which we have presented it in this chapter in the context of the ATR CHATR system (Black and Taylor, 1994). The idea of automatically generating synthesis units by clustering was first invented by Nakajima and Hamada (1988), but

was developed mainly by (Donovan, 1996) by incorporating decision tree clustering algorithms from speech recognition. Many unit selection innovations took place as part of the ATT NextGen synthesizer (Syrdal et al., 2000; Syrdal and Conkie, 2004).

We have focused in this chapter on concatenative synthesis, but there are two other paradigms for synthesis: **formant synthesis**, in which we attempt to build rules which generate artificial spectra, including especially formants, and **articulatory synthesis**, in which we attempt to directly model the physics of the vocal tract and articulatory process.

Formant synthesizers originally were inspired by attempts to mimic human speech by generating artificial spectrograms. The Haskins Laboratories Pattern Playback Machine generated a sound wave by painting spectrogram patterns on a moving transparent belt, and using reflectance to filter the harmonics of a waveform (Cooper et al., 1951); other very early formant synthesizers include Lawrence (1953) and Fant (3951). Perhaps the most well-known of the formant synthesizers were the **Klatt formant synthesizer** and its successor systems, including the MITalk system (Allen et al., 1987), and the Klattalk software used in Digital Equipment Corporation's DECtalk (Klatt, 1982). See Klatt (1975) for details.

Articulatory synthesizers attempt to synthesize speech by modeling the physics of the vocal tract as an open tube. Representative models, both early and somewhat more recent include Stevens et al. (1953), Flanagan et al. (1975), Fant (1986) See Klatt (1975) and Flanagan (1972) for more details.

Development of the text analysis components of TTS came somewhat later, as techniques were borrowed from other areas of natural language processing. The input to early synthesis systems was not text, but rather phonemes (typed in on punched cards). The first text-to-speech system to take text as input seems to have been the system of Umeda and Teranishi (Umeda et al., 1968; Teranishi and Umeda, 1968; Umeda, 1976). The system included a lexicalized parser which was used to assign prosodic boundaries, as well as accent and stress; the extensions in Coker et al. (1973) added additional rules, for example for deaccenting light verbs and explored articulatory models as well. These early TTS systems used a pronunciation dictionary for word pronunciations. In order to expand to larger vocabularies, early formant-based TTS systems such as MITlak (Allen et al., 1987) used letter-to-sound rules instead of a dictionary, since computer memory was far too expensive to store large dictionaries.

Modern grapheme-to-phoneme models derive from the influential early probabilistic grapheme-to-phoneme model of Lucassen and Mercer (1984), which was originally proposed in the context of speech recognition. The widespread use of such machine learning models was delayed, however, because early anecdotal evidence suggested that hand-written rules worked better than e.g., the neural networks of Sejnowski and Rosenberg (1987). The careful comparisons of Damper et al. (1999) showed that machine learning methods were in generally superior. A number of such models make use of pronunciation by analogy (Byrd and Chodorow, 1985; ?; Daelemans and van den Bosch, 1997; Marchand and Damper, 2000) or latent analogy (Bellegarda, 2005); HMMs (Taylor, 2005) have also been proposed. The most recent work makes use of joint **graphone** models, in which the hidden variables are phoneme-grapheme pairs and the probabilistic model is based on joint rather than conditional likelihood (Deligne et al., 1995; Luk and Damper, 1996; Galescu and Allen, 2001; Bisani and Ney, 2002;

Graphone

Chen, 2003).

Fujisaki

There is a vast literature on prosody. Besides the ToBI and TILT models described above, other important computational models include the **Fujisaki** model (Fujisaki and Ohno, 1997). IViE (Grabe, 2001) is an extension of ToBI that focuses on labelling different varieties of English (Grabe et al., 2000). There is also much debate on the units of intonational structure (**intonational phrases** (Beckman and Pierrehumbert, 1986), **intonation units** (Du Bois et al., 1983) or **tone units** (Crystal, 1969)), and their relation to clauses and other syntactic units (Chomsky and Halle, 1968; Langendoen, 1975; Streeter, 1978; Hirschberg and Pierrehumbert, 1986; Selkirk, 1986; Nespor and Vogel, 1986; Croft, 1995; Ladd, 1996; Ford and Thompson, 1996; Ford et al., 1996).

Intonation unit

Tone unit

HMM synthesis

One of the most exciting new paradigms for speech synthesis is **HMM synthesis**, first proposed by Tokuda et al. (1995b) and elaborated in Tokuda et al. (1995a), Tokuda et al. (2000), and Tokuda et al. (2003). See also the textbook summary of HMM synthesis in Taylor (2008).

More details on TTS evaluation can be found in Huang et al. (2001) and Gibbon et al. (2000). Other descriptions of evaluation can be found in the annual speech synthesis competition called the **Blizzard Challenge** (Black and Tokuda, 2005; Bennett, 2005).

Blizzard Challenge

Much recent work on speech synthesis has focused on generating emotional speech (Cahn, 1990; Bulut1 et al., 2002; Hamza et al., 2004; Eide et al., 2004; Lee et al., 2006; Schroder, 2006, inter alia)

Two classic text-to-speech synthesis systems are described in Allen et al. (1987) (the *MITalk* system) and Sproat (1998b) (the Bell Labs system). Recent textbooks include Dutoit (1997), Huang et al. (2001), Taylor (2008), and Alan Black's online lecture notes at `http://festvox.org/festtut/notes/festtut_toc.html`. Influential collections of papers include van Santen et al. (1997), Sagisaka et al. (1997), Narayanan and Alwan (2004). Conference publications appear in the main speech engineering conferences (INTERSPEECH, *IEEE ICASSP*), and the *Speech Synthesis Workshops*. Journals include *Speech Communication*, *Computer Speech and Language*, the *IEEE Transactions on Audio, Speech, and Language Processing*, and the *ACM Transactions on Speech and Language Processing*.

Exercises

8.1 Implement the text normalization routine that deals with MONEY, i.e. mapping strings of dollar amounts like *$45*, *$320*, and *$4100* to words (either writing code directly or designing an FST). If there are multiple ways to pronounce a number you may pick your favorite way.

8.2 Implement the text normalization routine that deals with NTEL, i.e. seven-digit phone numbers like *555-1212*, *555-1300*, and so on. You should use a combina-

tion of the **paired** and **trailing unit** methods of pronunciation for the last four digits. (Again you may either write code or design an FST).

8.3 Implement the text normalization routine that deals with type DATE in Fig. 8.1.2

8.4 Implement the text normalization routine that deals with type NTIME in Fig. 8.1.2.

8.5 (Suggested by Alan Black). Download the free Festival speech synthesizer. Augment the lexicon to correctly pronounce the names of everyone in your class.

8.6 Download the Festival synthesizer. Record and train a diphone synthesizer using your own voice.