
An Overview of Speech Synthesis and Recognition

1.1 Introduction

Speech processing comes as a front end to a growing number of language processing applications. We already saw examples in the form of real-time dialogue between a user and a machine: voice-activated telephone servers, embedded conversational agents to control devices, i.e., jukeboxes, VCRs, and so on. In such systems, a speech recognition module transcribes the user's speech into a word stream. The character flow is then processed by a language engine dealing with syntax, semantics, and finally by the back-end application program. A speech synthesizer converts resulting answers (strings of characters) into speech to the user. Figure 1.1 shows how speech processing is located within a language processing architecture, here to be a natural language interface to a database.

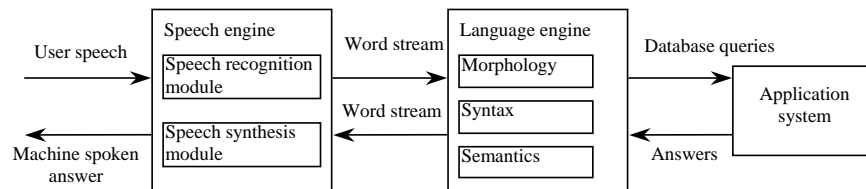


Fig. 1.1. Speech recognition and synthesis front ends.

Speech recognition is also an application in itself, as with speech dictation systems. Such systems enable users to transcribe speech into written reports or documents, without the help (pain?) of a keyboard. Most speech dictation systems have no other module than the speech engine and a statistical language model. They do not include further syntactic or semantic layers.

Speech processing for synthesis as well as for recognition involves techniques somewhat different from those we have already used in this book, namely a high

volume of mathematics in the form of signal processing and statistics. In this chapter, we will examine essential issues while trying to keep the material legible.

1.2 Some Basics of Phonetics

1.2.1 Sound Capture

Multimedia computers and signal processing are the basic instruments of modern phonetic studies. In addition to a computer, a digital sound card and a microphone are equipment we need to capture speech and to visualize waveforms – signals. Digital signal processing is the fundamental technique that enables us to modify the waveform representation and to ease the description of speech properties.

Many sound cards sample signals at a frequency of 44.1 kHz, which is that of compact disks. That means that the sound card captures a speech element 44,100 times per second. An analog-to-digital converter (ADC) digitizes the amplitude (volume) of each sampled element onto a digital scale. Amplitude values range from 0 to a maximum that depends on the encoding. Common encoding uses two bytes or 16 bits that can represent 65,536 amplitude values (Fig. 1.2).

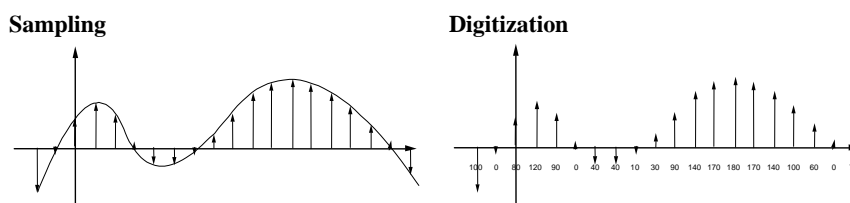
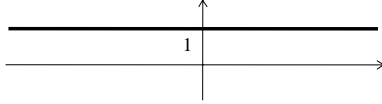
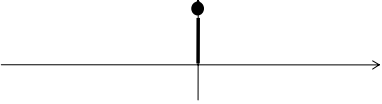
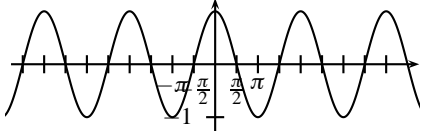
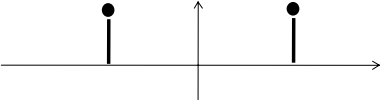

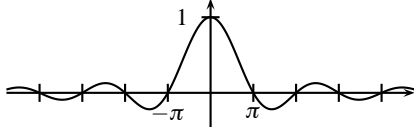


Fig. 1.2. Sampling and digitization of signals.

Digital sound synthesis is the reverse operation. It is carried out using the digital-to-analog converter (DAC) of the sound card that maps digital codes onto analog signals. Sounds can be amplified and made audible by a loudspeaker. Sound file formats might be different for different operating systems or software. Sounds can be stored and played back using, for example, .wav or .au files. Sound recordings can be digitized in other formats and compressed to save storage.

At the end of the 18th century, Fourier showed that any waveform – including speech – is a sum of sinusoids, each with a definite frequency. A discrete Fourier transform and its computerized optimization, the fast Fourier transform (FFT), are methods to decompose digital time signals into their corresponding sinusoids and hence to map a signal onto its composing frequencies. The frequency set of a signal is also called the Fourier spectrum. Table 1.1 shows examples of Fourier transforms for some functions.

Table 1.1. Fourier transforms for some functions.

Time domain	Frequency domain (Fourier Transforms)
Unit constant function: $f(x) = 1$ 	Delta function, perfect impulse at 0: $\delta(x)$ 
Cosine: $\cos(2\pi\omega x)$ 	Shifted deltas: $\frac{\delta(x+\omega)+\delta(x-\omega)}{2}$ 
Square pulse : $w_a(x) = \begin{cases} 1 & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0 & \text{elsewhere} \end{cases}$ 	$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ 

1.2.2 Phonemes

Frequency analysis is a more natural observation tool for speech than time analysis. However, the spectrum of a complete utterance, such as a sentence, is not of interest. Relevant acoustic properties are attached to sound units that show relatively stable properties over a time interval. A sentence or even a word span are obviously too large a unit. This was noticed as early as the end of the 19th century, well before automated spectrum analysis. Phoneticians then invented the concept of phoneme to delimit elementary speech segments.

Phonemes are a finite set of abstract symbols that are used to annotate real sounds: the **phones**. Inside a same phoneme class, phones show acoustic variations since speakers never pronounce a phoneme exactly the same way. The **allophones** are the members of the phone collection represented by a same phoneme.

Phonemes are not universal but are specific to a language. Two different phones may belong to a single phoneme class in one language and to two different classes in another. An English, a French, and a German *r* correspond obviously to different sounds. However, most native speakers would recognize them as the same phoneme, an *r*. They are thus allophones of a same phoneme. On the contrary, *leave* /li:v/ and *live* /lɪv/ are different words in English because pronunciations of /i:/ and /ɪ/ are perceived as different phonemes. This is not true in French, and many French

native speakers have trouble distinguishing between both words because sounds corresponding to /i:/ and /i/ are allophones of a same phoneme in this language.

Phonemes of many languages in the world have been itemized and have received a standard symbol from the International Phonetic Association (IPA), typically 40–50 or so for European languages. Phonemes are classically divided into vowels and consonants. Vowels /æ/ as in *sat*, /ʌ/ as in *come*, /ʊ/ as in *full* can be pronounced alone. Consonants need an additional vowel to be uttered: /eɪ/, /em/, /en/, /pi:/ (pee). Phonetic symbols enable us to annotate fragments of speech as well as words of a dictionary to indicate their pronunciation.

A phoneme's spectrum yields its acoustical structure. To obtain it, we could record isolated vowels or, for consonants, two adjacent phonemes – diphonemes – and carry out a Fourier transform on them; but this wouldn't be real speech conditions. So a better idea is to record a complete utterance and to carry out a FFT over short intervals or **frames**. The frame size is selected so that signal properties are stable – quasi-stationary – in it. Phoneme duration depends on the speaking rate, roughly 40 to 160 ms. Typically frames span 20 ms and overlap by 10 ms (Fig. 1.3). A sufficient sampling rate for speech analysis is 16 kHz, which corresponds to 320 speech samples per frame.

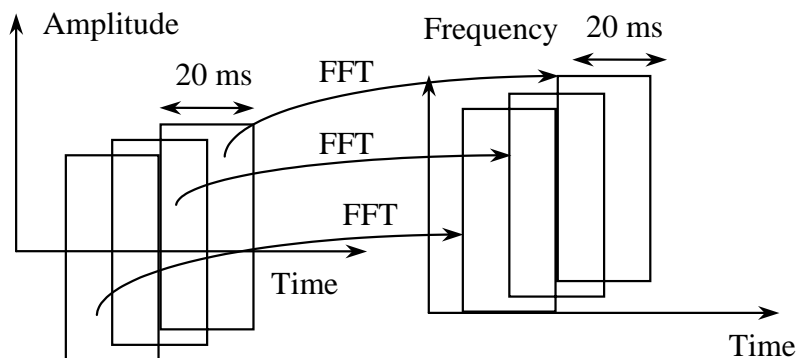


Fig. 1.3. The principles of a speech spectrogram: a FFT is carried out on time waveforms spanning 20 ms and mapped onto the sequence of spectra.

Using a spectrogram, we can then observe the frequency properties of an utterance over time and the sequence of phonemes composing it. Figure 1.4 displays a time waveform corresponding to the utterance

The boys that I saw yesterday morning

and below its spectrogram.

Vowels. Frequency features of a vowel include its fundamental frequency, the **pitch**, also denoted F_0 , and its “harmonics,” the **formants**. The pitch is directly related to the vibration frequency of the vocal cords in the throat. Although musical harmonics

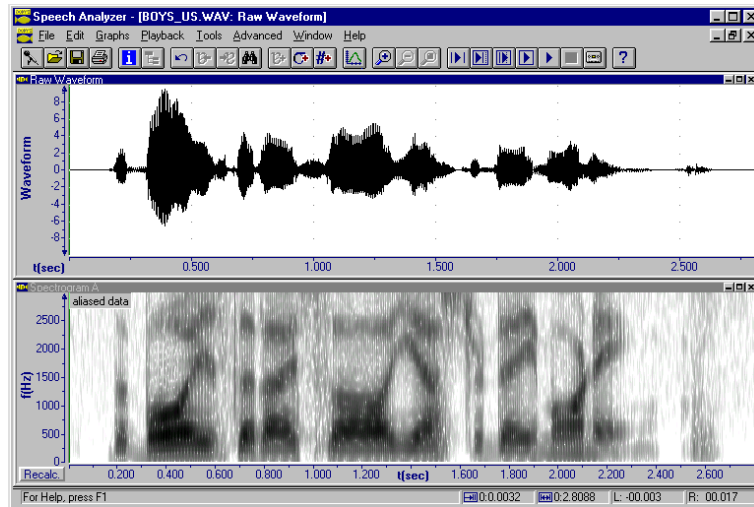


Fig. 1.4. Waveform and spectrogram of *The boys that I saw yesterday morning* obtained with the Speech Analyzer 1.5 program from JAARS Inc.

are multiples of a fundamental frequency, formants do not necessarily correspond to exact multiples of the pitch. Formants merely combine harmonics of the pitch with resonance properties of the vocal apparatus: cavities, tongue, lips, etc.

Each vowel has typical values of formants (Table 1.2), which are visible on the spectrogram. They correspond to the darker horizontal stripes that sometimes go up or down. Phonetic studies consider mainly the two first formants, F1 and F2, because they enable us to discriminate vowels and others formants are much less audible. However a third and fourth formants, F3 and F4, are also visible. Among other features, loudness refers to the intensity of phonemes and **timbre** to the intensity of formants.

The pitch value is specific to each individual. This value corresponds to the natural tone of our voice. Men have a pitch value located somewhere around 120 Hz, and women around 220 Hz. From its average value, the pitch varies during an utterance as people can change their voice from deep to high-pitched. Typically the pitch value of a man can range from 50 Hz to 250 Hz, and for a woman, from 120 Hz to 480 Hz.

Table 1.2. Average formant locations of vowels of American English. After Peterson and Barney (1952), cited by Deller Jr. et al. (2000).

Formants (Hz)	/i:/	/ɪ/	/e/	/æ/	/ɑ/	/ɔ/
F1	270	390	530	660	730	570
F2	2290	1990	1840	1720	1090	840
F3	3010	2550	2480	2410	2440	2410

Voicing refers to the vibration of vocal cords. Vowels are the best examples of voiced sounds, and spectrograms help track their periodic structure. Figure 1.5 shows the English diphthong /ɔɪ/ between 0.350 s and 0.600 s. It starts as an /ɔ/ and terminates as an /ɪ/. The formant pair (F1, F2) evolves accordingly and goes from (~ 550 Hz, ~ 850 Hz) to (~ 400 Hz, ~ 2000 Hz). The vowels can also be classified according to the tongue position in the mouth.

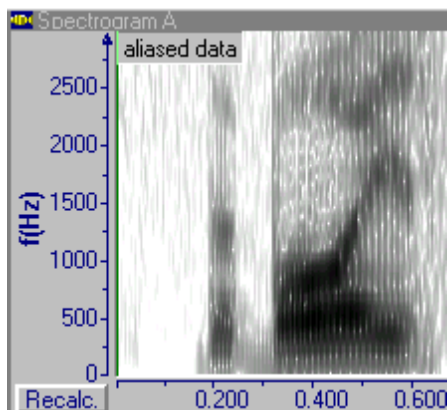


Fig. 1.5. Spectrogram of *The boys*.

Consonants. In contrast with vowels, consonants have a more even power distribution over the frequency spectrum. An ideally constant distribution over the frequency range is termed a white noise in signal processing. White noise sounds like the *shhhhh* of an unplugged loudspeaker. We can now understand some consonants as types of noise.

Consonants obstruct the airflow, and their distinctive features come from modifications they undergo in the mouth and from a possible periodic vibration. Some consonants are just similar to sorts of noise and are said to be unvoiced: /p/, /t/, /k/, /f/, /s/. Others involve vibrations of the vocal cords and are said to be voiced: /b/, /d/, /g/, /v/, /z/.

Consonants are classified using two parameters: the place and the manner of obstruction. Table 1.3 shows the classification American English consonants.

Consonants do not show a completely stable pattern in spectrograms. They are subject to modifications according to their left and right context, i.e., their neighboring phonemes. This phenomenon is called **coarticulation**, and it is the consequence of two overlapping articulations. Figure 1.6 shows an example of coarticulation in the imaginary word *keekoo* /ki:ku:/ where /k/ of diphoneme /ki:/ has a different pattern from diphoneme /ku:/.

Narrow and Broad Transcription. In fact, the two letters *k* in *keekoo* /ki:ku:/ correspond to two different classes of allophones. We can represent them more pre-

Table 1.3. Place and manner of articulation for American English consonants. Places correspond to *columns* and include the lips (labial), teeth (dental), ridge (alveolar), palate (palatal), velum (velar), and the vocal cords (glottal). Manners correspond to *rows* and include constricting (fricative) or blocking (plosive) the airflow, lowering the velum (nasal), narrowing (approximant). After the International Phonetic Association (1999).

	Labial	Labio-dental	Dental	Alveolar	Post-alveolar	Palatal	Velar	Glottal
Plosive	p b			t d			k g	ʔ
Affricate					tʃ dʒ			
Nasal	m			n			ŋ	
Fricative		f v	θ ð	s z	ʃ ʒ			h
Approximant				r		j	w	
Lateral approximant					l			

cisely using a finer transcription that will show a difference between these two variants. In this context, the representation we have used so far where a phoneme is written between slashes as /k/ is said to be a phonemic or broad transcription. The narrow transcription refers more closely to sounds, the underlying articulation, and uses square brackets [k]. It describes allophone classes using more phonetic symbols and additional diacritic signs as [ci:kʉ:] for *keekoo*, where [c] designates a palatal /k/ and [k] a velar one. The allophone classes are then members of a set as for /r/ = {[r], [R], [ʁ], [ɹ], ...}.

The phonemic transcription is the one we find in dictionaries. It corresponds to a lexical viewpoint. Phoneticians often use the narrow transcription, which is more detailed, to transcribe what has really been pronounced. This is a general rule, however. In the official *Handbook of the International Phonetic Association* (1999), depending on the language, authors do not use the two notations exactly in the same way.

Allophonic variation due to coarticulation is to some extent predictable and can be modeled using **phonological rules**. Chomsky and Halle (1968) is a classical work for English that introduced a rule notation used in many other contexts such as morphology (see Chap. ??). A phonological rule specifies a phoneme variation given a right and left context.

1.2.3 Phonemes of English, French, and German

Tables 1.4–1.6 show the phonetic alphabets used for English, French, and German. Classically, the vowel class divides into monophthongs – simple vowels – and diphthongs – double vowels. The consonant class divides into obstruents (plosives, affricates, and fricatives) and sonorants (nasals, liquids, and semivowels).

In addition to the IPA symbols that require a specific font, several coding schemes using standard ASCII have been proposed to encode phonetic symbols. Among them, the ARPAbet from the DARPA for US English is the most famous. Speech Assess-

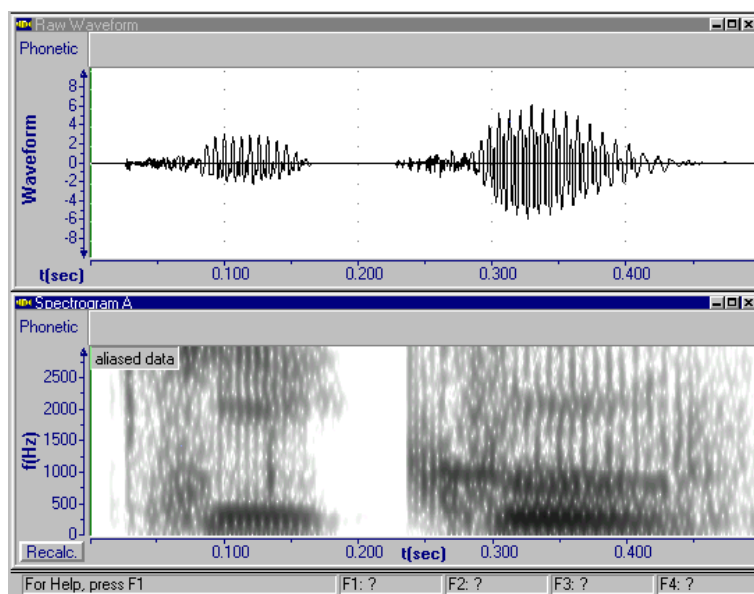


Fig. 1.6. Waveform and spectrogram of /ki:ku:/.

ment Methods – Phonetic Alphabet (SAMPA) is another attempt aimed at multilingual coverage.

The phonetic annotation can also describe **suprasegmental features**, i.e., speech characteristics that extend over more than one phoneme or are independent of it. The stress is an example of it because it applies to a syllable and to a vowel or a consonant. Table 1.7 shows the suprasegmental features of the International Phonetic Alphabet.

1.2.4 Prosody

Prosody corresponds to the melody and rhythm of speech. It accounts for a significant part of the intelligibility and naturalness of sentences. Prosody conveys syntactic, semantic, as well as emotional information. Prosodic aspects are often divided into **features** such as in English stress and intonation. Stress is a shorter-term variation that highlights a specific syllable or a semantically important word. Intonation is a longer-term variation that is linked to the grammatical structure. For instance, it applies differently to questions and declarations.

Prosodic features are roughly comparable to suprasegmental features. The pitch, loudness, and quantity are among the most notable ones. They correspond to physical properties, respectively the fundamental frequency, the intensity (or amplitude), and the duration. Perceptual and physical features are often confused. The relations between them are not trivial, however.

Prosodic features extend over a sentence, a phrase, and a word syllable. Each feature has specific parameters and a distinct model according to the information it

Table 1.4. US English phonetic alphabets. IPA and ARPAbet symbols.

Symbols		Examples	Symbols		Examples	Symbols		Examples
IPA	ARPAbet		IPA	ARPAbet		IPA	ARPAbet	
Vowels			Semivowels and liquids			Plosives		
[i:]	[iy]	<i>beat</i>	[l]	[l]	<i>let</i>	[p]	[p]	<i>pop</i>
[ɪ]	[ih]	<i>bit</i>	[r]	[r]	<i>red</i>	[b]	[b]	<i>bob</i>
[ɛ]	[eh]	<i>bet</i>	[j]	[y]	<i>yet</i>	[t]	[t]	<i>tot</i>
[æ]	[ae]	<i>bat</i>	[w]	[w]	<i>wet</i>	[d]	[d]	<i>dad</i>
[i]	[ix]	<i>roses</i>	Nasals			[k]	[k]	<i>kick</i>
[ə]	[ax]	<i>the</i>	[m]	[m]	<i>mom</i>	[g]	[g]	<i>gag</i>
[ʌ]	[ah]	<i>but</i>	[n]	[n]	<i>non</i>	[ʔ]	[q]	<i>bat</i> (glottal stop)
[u:]	[uw]	<i>boot</i>	[ŋ]	[ng]	<i>sing</i>	Closures of plosives		
[ʊ]	[uh]	<i>book</i>	Affricates			[b̚]	[bcl]	<i>voiced</i>
[ɔ]	[ao]	<i>bought</i>	[tʃ]	[ch]	<i>church</i>	[p̚]	[pcl]	<i>voiceless</i>
[ɑ]	[aa]	<i>cot</i>	[dʒ]	[jh]	<i>judge</i>	Flaps		
[ɜ]	[er]	<i>bird</i>	Fricatives			[ɾ]	[dx]	<i>butter</i>
[ə]	[axr]	<i>butter</i>	[f]	[f]	<i>fief</i>	[ɹ]	[nx]	<i>winner</i>
Diphthongs			[v]	[v]	<i>very</i>	Syllabics		
[eɪ]	[ey]	<i>ba<u>i</u>t</i>	[ð]	[dh]	<i>the<u>y</u></i>	[ŋ]	[en]	<i>bu<u>t</u>ton</i>
[aɪ]	[ay]	<i>bi<u>t</u>e</i>	[θ]	[th]	<i>th<u>i</u>ef</i>	[m]	[em]	<i>bot<u>t</u>om</i>
[ɔɪ]	[oy]	<i>bo<u>y</u></i>	[s]	[s]	<i>si<u>s</u></i>	[l]	[el]	<i>ba<u>t</u>tle</i>
[aʊ]	[aw]	<i>bo<u>u</u>gh</i>	[z]	[z]	<i>zo<u>o</u></i>	[ŋ]	[eng]	<i>Wa<u>sh</u>ington</i>
[oʊ]	[ow]	<i>bo<u>o</u>t</i>	[ʃ]	[sh]	<i>sh<u>o</u>e</i>	Others		
			[ʒ]	[zh]	<i>lei<u>s</u>ure</i>	[sil]	[sil]	<i>si<u>l</u>ence</i>
			[h]	[hh]	<i>ha<u>y</u></i>			

carries. Features are also specific to a language, although they exhibit many universal properties due to the common anatomy of all human beings. Sometimes prosodic features interact intricately in a way that has not been completely elucidated.

The sentence level intonation governs affirmations, exclamations, questions, and other grammar or discourse patterns. *Yes/no* questions such as *Is it correct?* requiring *yes* or *no* as an answer are uttered with a raising voice. Other questions such as *What do you want?* have a more constant tone. Pitch variation along the utterance is the main component of this level.

Pitch characteristics of questions are common to English, French, and German. Other pitch patterns are often different according to languages. For French, Delattre (1966b) has itemized ten types of intonation, which are shown in Fig. 1.7. Pierrehumbert (1980) is a frequently cited and comparable study for English.

Table 1.5. Phonetic alphabet used for French; IPA symbols.

Symbols	Examples	Symbols	Examples	Symbols	Examples
Vowels		Semivowels and liquids		Plosives	
[i]	<i>il</i>	[j]	<i>yeux</i>	[p]	<i>père</i>
[e]	<i>blé</i>	[w]	<i>oui</i>	[t]	<i>terre</i>
[ɛ]	<i>lait</i>	[ʁ]	<i>huile</i>	[k]	<i>cou</i>
[a]	<i>plat</i>	[l]	<i>lent</i>	[b]	<i>bon</i>
[ɑ]	<i>bas</i>	[ʀ] or [ʁ]	<i>rue</i>	[d]	<i>dans</i>
[ɔ]	<i>mort</i>			[g]	<i>gare</i>
[o]	<i>mot</i>	Nasals		Fricatives	
[u]	<i>genou</i>	[m]	<i>main</i>	[f]	<i>feu</i>
[y]	<i>rue</i>	[n]	<i>nous</i>	[s]	<i>sale</i>
[ø]	<i>peu</i>	[ɲ]	<i>agneau</i>	[ʃ]	<i>chat</i>
[œ]	<i>peur</i>	[ŋ]	<i>camping</i>	[v]	<i>vous</i>
[ə]	<i>le</i>			[z]	<i>zéro</i>
Nasalized vowels				[ʒ]	<i>je</i>
ẽ	<i>matin</i>			[h]	<i>hop!</i>
ã	<i>sans</i>			[x]	<i>jota</i>
õ	<i>bon</i>			[ʁ]	<i>haricot</i>
œ̃	<i>lundi</i>				

The pitch curve can be approximated to a polygonal line or linear segments inside phrases, that is, noun groups and verb groups. Each linear pitch segment has a starting and an ending value and is superimposed to other phoneme parameters.

Respiration is also part of the phrase prosody. The speaker tends to make a pause to breathe when s/he reaches the end of a syntactic or a sense group. Pauses and speech rate (speed of speech production) determine the rhythm. Speech rate is not linear over a sentence but is merely constant within breath groups – groups of words delimited by short pauses. In addition, breath groups and pitch groups – groups of words showing a specific intonation pattern – are related.

Within phrases, some words are more accented – or stressed in English. Phrase stress is associated with meaning in English, where informative words are more emphasized. French accent is more grammatical and falls regularly on the final syllable of syntactic groups. Some words in French can also receive a specific accent to mark an insistence.

The accentuation occurs on certain syllables of words. Again, it is different in English and French. The English stress is a matter of intensity unlike French, which shows little variation of this parameter. French accent is basically a question of duration. Accented syllables have a duration that is, on average, the double of unstressed syllables (Delattre, 1966a). In English, the stress position is indicated in the lexicon. French accent falls systematically on the final syllable except when the speaker insists on a word. The accent is then on the first or second syllable.

Table 1.6. Phonetic alphabet used for German; IPA symbols.

Symbols	Examples	Symbols	Examples	Symbols	Examples
Vowels		Additional diphthongs		Fricatives	
[a]	<i>h<u>a</u>t</i>	[iɐ]	<i>T<u>i</u>er</i>	[f]	<i>F<u>o</u>lge</i>
[aː]	<i>B<u>a</u>hn</i>	[ye]	<i>T<u>ü</u>r</i>	[v]	<i>W<u>e</u>in</i>
[ɛ]	<i>M<u>e</u>than</i>	[øɐ]	<i>G<u>e</u>hör</i>	[s]	<i>G<u>a</u>sse</i>
[eː]	<i>B<u>e</u>et</i>	[ɛɐ]	<i>G<u>e</u>währ</i>	[z]	<i>S<u>o</u>nn<u>e</u></i>
[ɛ]	<i>f<u>ü</u>llen</i>	[ɛɐ]	<i>G<u>e</u>weh<u>r</u></i>	[ʃ]	<i>S<u>ch</u>aden</i>
[ɛː]	<i>w<u>ä</u>hlen</i>	[aɐ]	<i>J<u>a</u>hr</i>	[ʒ]	<i>G<u>e</u>nie</i>
[ə]	<i>H<u>a</u>ck<u>e</u></i>	[oɐ]	<i>T<u>o</u>r</i>	[ç]	<i>d<u>i</u>ch</i>
[ɪ]	<i>W<u>i</u>nd</i>	[uɐ]	<i>R<u>u</u>hr</i>	[x]	<i>l<u>a</u>ch<u>e</u>n</i>
[iː]	<i>L<u>i</u>ed</i>	Semivowels and liquids		[h]	<i>h<u>a</u>lt<u>e</u>n</i>
[o]	<i>M<u>o</u>ral</i>	[l]	<i>l<u>a</u>den</i>	Plosives	
[oː]	<i>r<u>o</u>t</i>	[ʀ]	<i>r<u>e</u>ich<u>e</u>n</i>	[p]	<i>pl<u>a</u>tt</i>
[ɔ]	<i>Or<u>t</u></i>	[ʁ]	<i>w<u>a</u>ren</i>	[b]	<i>B<u>i</u>r<u>n</u>e</i>
[œ]	<i>g<u>ö</u>ttlich</i>	[j]	<i>j<u>a</u>gen</i>	[t]	<i>Top<u>f</u></i>
[ʊ]	<i>u<u>n</u>d</i>	Nasals		[d]	<i>d<u>o</u>rt</i>
[uː]	<i>Bl<u>u</u>t</i>	[m]	<i>M<u>a</u>nn</i>	[k]	<i>K<u>a</u>rte</i>
[y]	<i>m<u>ü</u>ssen</i>	[n]	<i>n<u>e</u>in</i>	[g]	<i>g<u>e</u>hen</i>
[yː]	<i>R<u>ü</u>be</i>	[ŋ]	<i>g<u>i</u>ng</i>	Syllabics	
[ʏ]	<i>s<u>ü</u>ß</i>	[ɲ]	<i>K<u>o</u>gn<u>a</u>k</i>	[ŋ]	<i>h<u>a</u>tten</i>
[ø]	<i>Ö<u>ko</u>nom</i>	Affricates		[m]	<i>gr<u>o</u>ß<u>e</u>m</i>
[øː]	<i>Ö<u>l</u></i>	[pf]	<i>P<u>f</u>ahl</i>	[l]	<i>K<u>e</u>ss<u>e</u>l</i>
[ɐ]	<i>b<u>e</u>ss<u>e</u>r</i>	[ts]	<i>Z<u>a</u>hl</i>	[ɺ]	<i>H<u>a</u>ken</i>
Main diphthongs		[tʃ]	<i>d<u>e</u>utsch</i>		
[aɪ]	<i>E<u>i</u>s</i>	[dʒ]	<i>D<u>sch</u>ung<u>e</u>l</i>		
[aʊ]	<i>H<u>a</u>us</i>				
[ɔʏ]	<i>K<u>r</u>euz</i>				

Table 1.7. Suprasegmental features. After the International Phonetic Association (1999).

Symbols	Descriptions	Examples
ˈ	Primary stress	[ˈfoʊnəˈtʃən]
ˌ	Secondary stress	
:	Long	[eː]
ˑ	Half-long	[eˑ]
ˑ	Syllable break	[i.ækt]

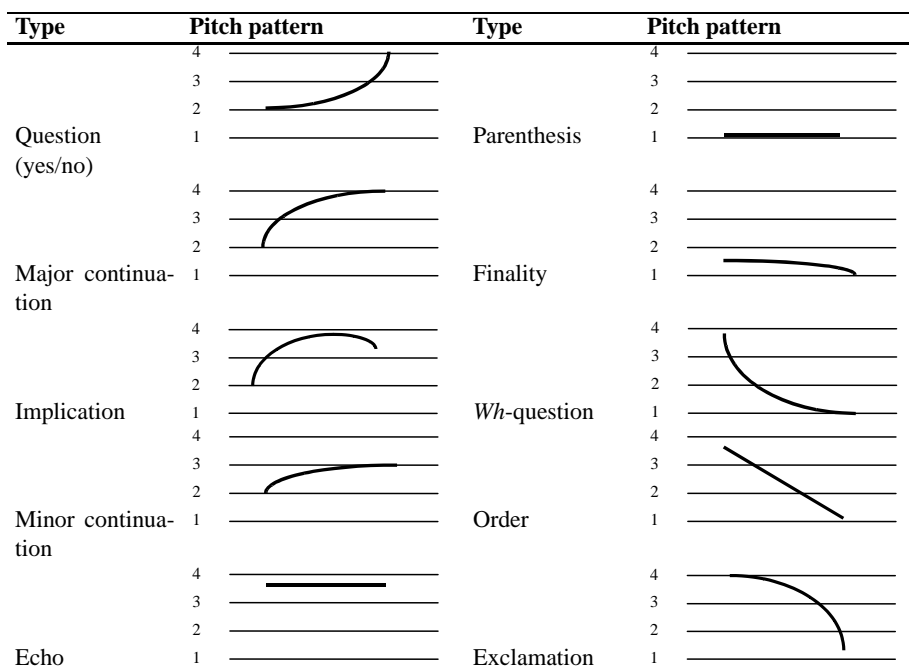


Fig. 1.7. The ten fundamental intonations of French (Delattre, 1966b, pp. 4–6).

1.3 Speech Synthesis

1.3.1 Rudimentary Techniques

A first rudimentary method to synthesize speech is to record complete or partial messages: welcoming announcements, whole questions, or answers. The application program, a speech server for instance, will play static messages when it needs them, possibly concatenating partial messages. Although this method is used in airports, railway stations, and in many other places, it does not prove very flexible. All messages must be recorded in advance to be uttered by a machine. If a message changes, it has to be rerecorded. This requires careful organization to be sure that the application program will not require a nonexistent message. However, recorded speech is sometimes preferred because it sounds more natural than other synthesis techniques.

A second method is to record all the words of the application lexicon in a digital format. To generate a spoken message, the synthesizer sequentially looks up the words in the lexicon, fetches their digital recordings, concatenates them, and converts the digital codes into sounds using the digital to analog converter of the sound card. This second method is a close extension of the first one. It requires recording all the words of the vocabulary, but it makes it more flexible for an application program to synthesize a message. The synthesizer is no longer constrained by fixed templates. However, the synthesized speech does not sound as natural. The concatenation of

words is never perfect and leaves some glitches on word boundaries. In addition, utterance of words varies according to prosodic context. So different pronunciations of the same word, corresponding to different prosodic features, would have to be recorded. If not impossible, this makes this kind of synthesis much more expensive.

1.3.2 Speech Synthesis with Phonemes

From whole messages and words, we can further shrink our sound synthesis units to phonemes. Instead of a database containing a recording of all the words, we simply need a sound recording of all the vowels or consonants. This dramatically scales down storage requirements. In addition, speech synthesis with phonemes enables us to deal with a potentially unlimited vocabulary and to generate any message dynamically. Phonemic speech synthesizers are also called text-to-speech converters (TTS) because they can read any text aloud.

At a first glance, speech synthesis of a message using phonemes is easy to implement. It would consist in concatenating sounds of all the phonemes (phones) making up the message, and in uttering them. In fact, coarticulation has a considerable influence that completely modifies the phone patterns. It makes a synthesized message using single phones sometimes hardly understandable.

A simple experiment with a sound editor can help us realize how we perceive coarticulation. Recording the “word” *keekoo* [ki:ku:] (Fig. 1.6) then cutting the phone [k] from diphone [ki:], to assemble it with phone [u:] of diphone, [ku:] yields a sound resembling [pu:] or [tʃu:]. It may seem paradoxical, but anyone can carry out this experiment on a computer equipped with a sound card. This explains why synthesis using a concatenation of single phones yields poor-quality results.

Two main strategies are available to cope with coarticulation. The first one is similar to the ones used in electronic music synthesizers. It uses a digital oscillator, noise generators, and frequency filters. It uses rules to model the modification of the formants while pronouncing the words. This technique requires a lofty phonetic knowledge, and specific rules are necessary for each language. It yields acceptable results when well implemented. This technique often has the favor of senior and knowledgeable phoneticians.

A second technique consists in recording all possible sound patterns, here the phoneme transitions, rather than modeling them. This technique is much more naïve than the first one and yet often has a superior quality. To implement it, we first collect a waveform for all the diphonemes of a given language. A speaker reads a text containing all the possible diphoneme combinations with a homogeneous voice and intonation. The recorded spoken text is then annotated with a sound editor, either manually or semiautomatically. A diphone is selected and extracted for each diphoneme. The diphones extend from the middle of the first phone to the middle of the second one. In addition, an instance of every phoneme is also isolated and stored to constitute the database.

In French, there are 39 phonemes. This produces the reasonable maximum figure of 1521 diphonemes (39^2), many of them never occurring in reality. To synthesize a text, the synthesizer splits the word stream into the sequence of diphonemes that

composes it. Then, the synthesizer looks in the database for the corresponding diphones and concatenates them. Single phones replace word beginnings and endings. For example, *Paris* [pæris] pronounced in English uses the diphone sequence: #P, PA, AR, RI, IS, and S#.

In practice, in addition to diphones, the current trend in speech synthesis is to use large databases of sound patterns to include more polyphones: triphones, quadri-phones, etc. Using phonetic units of variable length has enabled a dramatic progress in speech synthesis quality in recent years. This made it possible to combine the quality of prerecorded messages with the flexibility of diphone synthesis. For a pioneering work implemented in commercial systems and applications, see Le Meur (1996).

To have a flexible synthesis or to implement prosodic patterns, it is necessary to adjust suprasegmental features: the phone duration, intensity, and fundamental frequency (pitch value), as in Table 1.8.

Table 1.8. Diphones with duration in milliseconds, intensity, and pitch in Hz. The pitch value is defined at the beginning of the diphone. The pitch curve is interpolated as linear segments between diphones. Naturally, pitch applies only to vowels and voiced consonants.

	Diphones	Duration	Intensity	Pitch
#P	#[p]	70	80	120
PA	[pæ]	100	80	180
AR	[ær]	100	70	140
RI	[ri]	70	70	120
IS	[is]	70	60	100
S#	[s]#	70	60	80

Since original diphones have probably been recorded with different pitch and duration values, the speech synthesizer must modify them before the digital-to-analog conversion. This is achieved by techniques such as pitch-synchronous overlap add (PSOLA) (Moulines and Charpentier, 1990).

The PSOLA technique splits the voiced parts of the diphones into a sequence of frames. The size of each frame is two pitch periods and is obtained by multiplying the original waveform by a Hamming window. The pitch is detected from the waveform, and the windows are centered on the pitchmarks corresponding to the closure of the glottis. Repositioning the pitchmarks at the desired pitch value creates a diphone with a new pitch value. Closer pitchmarks produce a higher pitch, while spacing the pitchmarks at a greater distance produces a lower pitch. Finally, the frames are overlapped and added to produce the new diphone. PSOLA has the advantage of being an efficient algorithm that is able to carry out the pitch modifications in real time.

1.3.3 Text Processing for Speech Synthesis

Pronunciation of a word does not always correspond to its spelling, and a text synthesizer requires a module mapping a text's letters onto phonemes. Phonetic transcription is regular in languages like Spanish, where a word spelling often gives sufficient hints to pronounce the word properly. The letter-to-sound conversion can then be handled by a small set of transcription rules. In other languages, like English, a lexicon is needed due to a large number of exceptions. Consider, for instance, the letter *i* in *give* [gɪv] and in *life* [laɪf]. Other similar examples are countless; this does not even include the stress accent, which is often random and sometimes different between US and British English.

A text-to-speech program is usually organized in several stages:

1. *Tokenization*. Sentence separation, punctuation processing, and word breaking.
2. *Lexical access and morphological analysis*. A morphological parsing splits words into morphemes: lemma and affixes. Search of the words in a dictionary to process the exceptions. Some morphological rules may lead to an irregular pronunciation, as in English. Compare *played and worked* with *rugged and ragged*.
3. *Grammatical analysis*. This stage is not always implemented. A grammatical module can carry out a part-of-speech tagging, a group detection, or a full syntactic parsing, depending on the desired performance. The grammatical module removes part-of-speech ambiguities that may have a consequence on the pronunciation and enables the synthesizer to generate prosodic features.
4. *Phonetic translation*. Transcription rules process the words supposed to be regular. These rules generally consider right and left contexts of a letter to produce its phonetic transcription.

1.3.4 Letter-to-Phoneme Rules

The current tendency to produce a phonetic transcription of a text is to use big pronunciation dictionaries, where the synthesizer looks up the individual transcription of each word of text. However, a part of the phonemic transcription still depends on rules, notably to deal with unknown words or with proper nouns. Depending on the irregularity of a language, these rules are simple or more complex to write, but they follow the same formalism.

Transcription rules translate letters into a sound given constraints on their left and right neighbors. Letter-to-sound mapping may concern one single letter, as for *k* and *s* in *keys* [ki:z], or a string of two or more letters, as for *ey* or for *qu* and *ay* in *quay* [ki:]:

$$\frac{k \quad ey \quad s \qquad \quad qu \quad ay}{[k] \quad [i:] \quad [z] \qquad \quad [k] \quad [i:]}$$

In the context of text-to-phoneme conversion, strings of letters mapped onto a phoneme are generally called **graphemes**.

The transcription rule format is the same as what we saw with morphological processing (Chap. ??). The rule

$$X \rightarrow [y] / \langle lc \rangle _ \langle rc \rangle$$

means that grapheme X is mapped onto the phone [y] under the constraint of a left context <lc> and a right context <rc>, both being graphemes:

$$\begin{array}{c} \text{Text} \\ \hline \hline \text{Phonetic transcription} \quad \dots \quad [y] \quad \dots \end{array} \quad \begin{array}{c} \langle lc \rangle \quad X \quad \langle rc \rangle \end{array}$$

Rules may have no constraint on their left or right context, such as the rules

$$X \rightarrow [y] / _ \langle rc \rangle$$

$$X \rightarrow [y] / \langle lc \rangle _$$

or may be context-free, such as the rule

$$X \rightarrow [y]$$

Such a rule format is derived from the *Sound Pattern of English* formalism (Chomsky and Halle, 1968).

As a simplified example, modified from Divay and Vitale (1997), let us model the pronunciation of the letter *c* in English. Let us suppose it is pronounced either [s] before *e*, *i*, or *y*, or [k] elsewhere. The rules governing the transcription are

$$c \rightarrow [s] / _ \{e, i, y\}$$

$$c \rightarrow [k] / _ \{a, b, c, d, f, g, h, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, z, \#\}$$

where # denotes the end-of-string symbol. The rules are usually tried from top to bottom. If the first one fails, the second one succeeds since the context of both rules complements each other. This means that no context is necessary in the second rule. The previous set could have been more compactly written as:

$$c \rightarrow [s] / _ \{e, i, y\}$$

$$c \rightarrow [k]$$

The transcription rules can handle the English stress in some cases such as for suffixes. Ending morpheme *ation*, as in *information*, is transcribed by

$$\text{ation} \rightarrow [1][eɪ] = [0][f][ə][n] / _ +$$

where 1 indicates a primary stress, 2 a secondary stress, 0 an unstressed syllable, and = a syllable boundary. The symbol “+” is a morpheme boundary mark (Divay and Vitale, 1997).

The transcription rules are certainly more intricate in English than in other European languages. A frequently cited example is *ough*, which has a very ambiguous pronunciation. Consider: *rough* [ʌf], *through* [u:], *bough* [aʊ], *though* [ɔ:], *dough* [əʊ], and *cough* [ɔ:f]. To give some idea of the complexity of transcription, the pioneering DECTalk speech synthesis program had a lexicon of 15,000 words and 1500 rules for English (Hallahan, 1996).

1.3.5 Text-to-Phoneme Transcription

The text-to-phoneme converter translates a text into a sequence of phonetic symbols. It uses the dictionary and the set of grapheme-to-phoneme rules. The converter applies rules with the longest match algorithm. That is, when several rules match a text string and are therefore potentially valid, the converter retains the one with the longest matching string. In the case when the rules have a left-hand-side symbol of a same length, they are tried from top to bottom until the converter finds a valid matching context (left and right contexts of the rule are true).

The conversion program processes the input text sequentially from left to right. A pointer is set on the next character to be processed. Initially the pointer is set at the beginning of the input text. The program selects the rule that has the longest match between its left-hand-side string and the text string starting from the pointer. If the left and right contexts match the text, the program applies the rule and converts the text string into phonemes. The size (span) of the translation buffer is that of the left-hand-side string of the selected rule. Once the string translation has been carried out, the pointer is moved forward to the next unprocessed character. If left and right contexts do not match, the program tries rules in decreasing length of their matching left-hand-side string.

As for morphological parsing, the phonetic transcription rules can be compiled into efficient finite-state transducers (Roche and Schabes, 1997).

1.3.6 Remaining Transcription Ambiguities

Grapheme-to-phoneme rules and a dictionary do not solve all problems. Sometimes the same word form has different pronunciations according to its part of speech or sense. A part-of-speech or sense tagging is then necessary to disambiguate more accurately all the word forms. Examples of pronunciation depending on part of speech include, in English, *use*, which has a different pronunciation whether it is a noun or a verb: *I use* [ju:z], but *a use* [ju:s]. Stress may also depend on category. Compare *to object* [ə:b'dʒekt], where the stress is on the second syllable, and *an object* ['abdʒɪkt] with a stress on the first syllable. In French, the sentence

Les poules du couvent couvent
‘Hens of the convent are brooding’

provides another example. The first *couvent* is a noun and is pronounced [kuvɑ̃]. The second one is a verb in the third-person plural and is pronounced [kuv]. More generally, the suffix *-ent* of French adverbs is pronounced [ɑ̃], while this same suffix is not pronounced in verbs of the first group, third-person plural.

Solving pronunciation ambiguities related to a part-of-speech membership requires a parser or a POS tagger. The transcription rules take the word annotation (adverb, noun, verb, ...) into account. The rule discarding the *-ent* suffix of French verbs reads:

ent/verb → ε / __ #

where # denotes the end-of-string symbol, and ϵ the null symbol.

In some other relatively rare contexts, the pronunciation depends on the sense. In English, compare *read* in *I read it yesterday* [rɛd] and in *I read it everyday* [ri:d]. Compare *deserts* in *You get your just deserts* [dɪ'zɜ:ts] and in *The deserts of Sudan* ['dezɜ:ts]. In French, compare *fil* in

Les fils de la nation [fis]
'The sons of the nation'

and

Les fils du destin [fil]
'The threads of destiny'

To solve the two last cases, *deserts* and *fil*, text-to-phoneme transcription can resort to a sense tagger. Yarowsky (1996) describes an algorithm based on unsupervised techniques comparable to the one described in Chap. ???. It annotates words with ambiguous pronunciations with their sense, and hence disambiguates English homographs such as *lead* ([led]/[li:d]), *bass* ([beɪs]/[bæs]), *wound* ([wu:nd]/[waʊnd]), dates from fractions, *5/16*, (*five-sixteenths* or *May 16th*), titles, *St.* (*Street* or *Saint*), etc. However, pronunciation ambiguities due to a particular sense are not that frequent, and a couple of rules of thumb applied to doubtful words are often sufficient.

1.3.7 Generating Prosody

Prosody synthesis involves the annotation of phrases with speech rate, pitch patterns, pauses, and phones with intensity and duration. Prosody is certainly the most difficult speech feature to reproduce. It is controlled by multiple factors, sometimes of a nonlinguistic nature. It is still a subject of active research. A distinctive metallic touch still betrays the imperfect implementation of prosody in most current artificial speech systems.

Annotation of a text with prosodic features requires parsing the input text either with a group detector, a chunker, or with a full parser before the phonetic transcription. Group detection is a first step to handle phenomena like junctures (transitions) between words. French *liaisons* linking two words are an example of it. *Liaisons* occur within breath groups and discard pauses between words. In addition, some letters, namely *s*, *z*, *r*, *t*, *d*, and *n*, which are usually silent when they are at the end of a word, are pronounced if the next word begins with a vowel. In a liaison, *s* is pronounced [z] as in

les petits ânes [leptizan]
'the little donkeys'

Breath groups roughly correspond to noun groups and verb groups. A group detector would annotate *les petit ânes* with tags such as

les/BeginNGroup petits/InsideNGroup ânes/EndNGroup.

and the transcription rules would use the group annotation to insert the liaison between two words

- → [z] / s/BeginNGroup __ Vowel
- → [z] / s/InsideNGroup __ Vowel

where - denotes a blank space, and `BeginNGroup` and `InsideNGroup` annotate the beginning and inside words of a noun group, respectively. Liaisons, however, are not systematic. Modeling them completely would require additional processing.

Transcription rules can help annotate phonetic strings with more complex prosodic features, such as a pitch pattern. This requires a full parse of the input text. For each sentence, the transcription rules have first to consider the parse result and select an appropriate pitch pattern within a predefined set, such that of Fig. 1.7. Then, rules apply the corresponding pitch curves to groups or to the whole sentence. Finally, curves are divided into linear segments and each phoneme is assigned with a pitch value. Boula de Mareüil et al. (2001) describe a complete implementation for French.

1.4 Speech Recognition

1.4.1 Defining Speech Recognition

Although it deals with the same raw material, speech, recognition is often stated as more complex than synthesis. Speech recognition complexity can be classified according to several factors:

- *Number of speakers.* Some devices can recognize speech from one single speaker and are then speaker-dependent. Others can recognize speech from any native speaker (speaker-independent). When a system is speaker-dependent, a user has to train it to her/his voice before the system can recognize her/him, often by reading a text of several pages. Speaker-independence is obtained by pretraining recognition systems with a large number of speakers, so when a new speaker talks to the system, s/he can expect to fall within already trained or modeled voice patterns.
- *Fluency of speech.* Older systems could recognize isolated words only, that is, the speaker had to pause between each word. Current systems operate with dictated continuous speech. Another technique – referred to as word spotting – is meant to recognize key words in a sequence of unknown words. Continuous speech recognition is more difficult because the combinatory to decode words from a phoneme string is increased. Compare, for instance: *an ice cream* and *a nice cream*. If there is a slight pause between each word, word decoding is naturally easier. Research is now focused on improving recognition performance of naturally flowing speech, that is, speech with hesitation or repairs.
- *Size of vocabulary.* Vocabulary is a limiting factor of many recognition devices. A larger vocabulary introduces more chances of confusion and requires more memory and computing resources. In fact, the size is not the only factor to take into

account. Many different words are easier to recognize than few similar words. Current commercial dictation systems reach vocabulary sizes surpassing 64,000 words, while digit recognition – small similar words – is still sometimes a problem.

- *Syntax.* A constrained syntax helps recognize words by disambiguating similar sounds. The average number of words that can follow a given word is also called the branching factor or perplexity. Given a word, a grammar reduces the perplexity. However, syntax often limits the domain of an application. In addition, sentences of naturally occurring speech contain speech repairs, which cannot be covered by classical syntactic formalisms.
- *Environment.* Many laboratories reported experiments with high recognition rates as early as 1980. Figures were then subject to interpretation. Environmental conditions accounted for much in these results. Very careful speakers were uttering words in insulated rooms. This yielded results that were not reproducible in real-world conditions. Real applications should provide telephone services or spoken control of machines, and these conditions make the figures decrease in a dramatic way. Current research includes the assessment of recognition of broadcast news, possibly with music in the background, or conversations.
- *Language.* Many laboratories working on speech recognition focus their research on English and optimize their tools for it. The research effort for English and, to a lesser extent, Western European languages (German, French, Spanish, Italian) is tremendously superior to that devoted to other languages. This means that systems available for English perform better. Recognition of other languages, even with a very large community of speakers like Arabic, Chinese, or Hindi, has for now a substantially lower quality.

In summary, it is easier to recognize a small number of isolated words from a single speaker in a quiet environment rather than real continuous speech through a telephone from anybody: you and me, possibly nonnative speakers.

1.4.2 The Structure of a Speech Recognition System

Speech recognition systems are usually split into several components that are based on mathematical considerations. Speech recognition is formulated in terms where a speaker utters a sequence of words

$$W = w_1, w_2, \dots, w_n.$$

The speaker transmits this information using a string of acoustic symbols. Let A be the sequence of symbols corresponding to W :

$$A = a_1, a_2, \dots, a_m,$$

where acoustic symbols are members of a finite alphabet. Although acoustic symbols and phonemes are tied, they are not to be confused. Next, we will describe how they are connected together.

The objective of a speech recognition system is to determine A and then to decode the most likely word sequence \hat{W} knowing A . The first part resorts to acoustic signal processing techniques, and the second one to linguistic modeling. The recognizer obtains the sequence of acoustic symbols A using an acoustic processor as a front end. It extracts parameters from speech frames and, depending on parameter values, labels each speech frame with an acoustic symbol (Fig. 1.8).

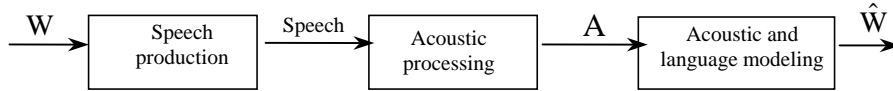


Fig. 1.8. The speech recognition process.

Word string decoding is then optimally formulated by

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(W|A).$$

Using Bayes' theorem,

$$P(W|A) = \frac{P(A|W)P(W)}{P(A)}.$$

$P(A)$ is the probability of the acoustic sequence. It is fixed for a given utterance and plays no role in the word sequence determination, which is a probability maximization. \hat{W} can then be rewritten in

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(A|W)P(W).$$

From this formula, we can divide word decoding into two distinct linguistic components that operate in parallel. The first one concerns the acoustic modeling of speech. Its goal is to optimize $P(A|W)$, that is, the acoustic string probability knowing that word string W has been uttered. The second one resorts to language models in order to optimize $P(W)$, that is, the word sequence probability.

Acoustic symbols do not map directly onto phonemes. A same phoneme can be realized by two different acoustic symbols. And conversely, in some contexts, two phonemes can be realized by a same acoustic symbol. Hidden Markov models (HMMs) are a mathematical device that enable us to decide which string of phonemes is behind a string of acoustic symbols (see Chap. ??).

The second component of word decoding uses probabilities on word sequences to find the most likely sequence uttered. This is also termed a statistical language model. Models are based on n -grams, that is, statistics on sequences of n contiguous words, and typically include unigrams, bigrams, trigrams, and even 4-grams (see Chap. ??).

Finally, the transcription of acoustic symbols into words attempts to find a string maximizing language and acoustic models. Trying all possible words to retain the

optimal sequence is not possible with large vocabularies. Therefore, in parallel to computing probabilities, the word decoder system carries out a hypothesis search and discards some candidates to reduce the search space.

1.4.3 Speech Parameters

Recognition devices do not process the waveform directly. Instead, they use signal processing techniques to derive a set of acoustic parameters from speech frames. Parameters should provide means of a decision, that is, to discriminate between phonemes and, as far as possible, be relatively easy to compute.

To have a discriminating potential, parameters should be related to “natural” features of speech such as whether a frame pertains to a voiced or unvoiced phone. Let us imagine a simple parameter giving a rough estimate of it. On spectrograms, voiced segments appear to be darker than unvoiced ones or silences. The darkness level of a frame can be computed by summing the gray level values of all pixels in the frame. This level visualizes the so-called energy of the frame: the darker the frame, the higher the energy. Although imperfect and in practice never used alone, energy is a parameter that when above a certain threshold enables us to classify roughly the frame as voiced.

Energy is easier to derive from a time waveform. Let us suppose that frame k , F_k , consists of N speech samples $s(m), s(m+1), \dots, s(m+N-1)$. The definition of the frame energy is:

$$E(F_k) = \sum_{n=m}^{m+N-1} s^2(n).$$

While energy is relatively easy to understand, other parameters involve a more elaborate signal processing knowledge. Common parameters used in speech recognition are derived from **linear prediction** coefficients. The idea behind linear prediction (LP) is that a speech sample is not independent from its predecessors. It can be approximated by a linear combination, where speech sample $s(n)$ is extrapolated from the m previous samples $s(n-1), s(n-2), \dots, s(n-m)$. The prediction equation is:

$$\hat{s}(n) = a(1)s(n-1) + a(2)s(n-2) + a(3)s(n-3) + \dots + a(m)s(n-m),$$

where $\hat{s}(n)$ is the predicted value, and $a(1), a(2), \dots, a(m)$ are the LP coefficients. Prediction coefficients are specific to each frame and are constant values over the frame. Prediction is made possible inside a frame because of the relatively slow transition of phones compared with the frame duration.

LP coefficients are estimated so that they result in the “best fit” to the real signal. This is often recast as a minimization of the mean squared error. The prediction error for the sample n is the difference between the predicted and the actual values:

$$e(n) = s(n) - \hat{s}(n),$$

and LP coefficients should minimize the summed squared error over a frame:

$$\sum_{n=k}^{k+N-1} e^2(n).$$

The precision of linear prediction depends on the number of coefficients. Typical numbers range from 10 to 14; beyond these numbers, LP does not show much improvement. Optimization methods, namely covariance and autocorrelation, enable us to solve this equation and estimate the best coefficients. Both methods are based on the differentiation of the summed squared error with respect to each coefficient.

As typical figures used in state-of-the-art speech recognition systems, speech is sampled at 16 kHz and parameter vectors are computed every 10 ms. It corresponds roughly to 5 to 10 vectors per phoneme. The frames span from 20 to 30 ms and are overlapping. Each frame is represented by vector of typically 40 parameters (Fig. 1.9). The parameters consist of the frame energy, a dozen LP coefficients, and their first- and second-order derivatives.

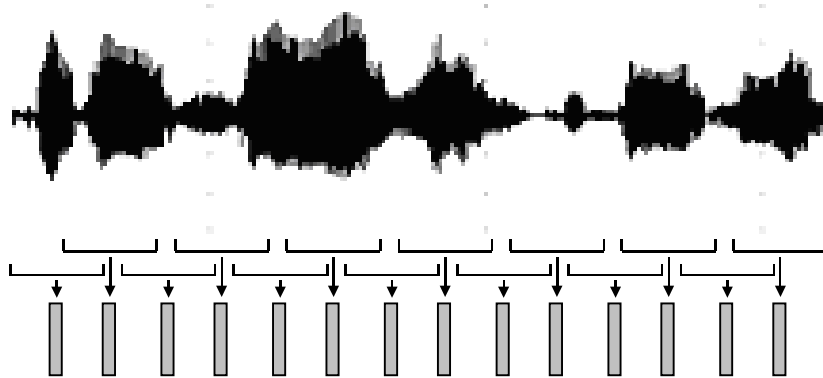


Fig. 1.9. The parameter vectors (in gray) are extracted from the speech signal (in black). The parameters are computed in each overlapping window.

Linear prediction coefficients are computed on the signal **cepstrum**, rather than on the raw waveform samples. The cepstrum is a sequence of a Fourier transform operated on the time samples, then a log transform, and finally an inverse Fourier transform (Fig. 1.10). The cepstrum shows better analysis properties.

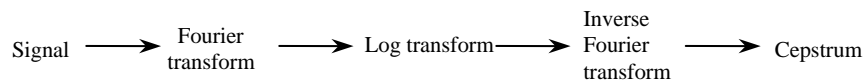


Fig. 1.10. The cepstrum.

Frame vectors are classified into a finite set of categories according to values of their parameters. Categories correspond to the acoustic symbols used to annotate

frames. Let us simplify the problem to understand the category's purpose, and let us suppose that the vectors can fall into two categories corresponding roughly to vowels and consonants – ω and χ . As a fictitious example, *assess* [ə'ses] could possibly produce the observation string $\omega\omega\omega\omega\omega\chi\omega\chi\chi\chi\chi\omega\omega\omega\omega\omega\chi\omega\chi\chi\chi\chi$, and *essay* ['eseɪ] $\omega\omega\omega\omega\omega\chi\chi\omega\chi\chi\chi\chi\omega\omega\omega\omega$. Ideally, the categories would exactly correspond to the phonemes. In fact, parameters are never perfect and, because of variations in the pronunciation, phonemes are sometimes annotated with a wrong symbol.

Because of the huge quantity of data, acoustic symbols are not crafted manually but are automatically obtained from speech corpora using classification techniques such as K -means. The set of symbols is called a **codebook**. Earlier systems used a codebook of 256 symbols. Present systems use raw frame vectors without a classification step. For the sake of simplicity, we will use the codebook notion from now on, first with two symbols ω and χ .

1.4.4 Acoustic Modeling: Hidden Markov Models

The signal processing stage transforms speech into a stream of acoustic symbols. The acoustic modeling component relates it to a string of phonemes or a word. Most speech recognition devices are based on HMMs to carry this stage out. We saw Markov models and the associated algorithms in Chap. ?? We will describe now how to apply them to words or phonemes.

When the recognition unit is a single word, the automaton is a network where a subchain represents each word of the vocabulary. The modeling network has a unique starting state and a unique final state that correspond to bounding silences surrounding the pronounced word. All the words of the vocabulary are linked to these states. Thus, there are as many branches as there are words. The acoustic symbol sequence provided by the acoustic processing unit will proceed along these branches and will receive a probability value for each branch. The recognized word corresponds to the highest value.

Each chain has a number of states that depends on the word length it represents (typically 30). Each state features a self-looping transition and a transition to the next state. The looping transitions model the phoneme durations, and other transitions model the passing from a phoneme to the next one. From then on, we will successively model a two-word vocabulary using plain automata, Markov chains, and finally Markov models.

1.4.5 Markov Chains

The automaton in Fig. 1.11 models the words *essay* ['eseɪ] and *assess* [ə'ses], respectively with three and four states. The states here are very simplified and are related either to a consonant emitting the acoustic symbol χ , or a vowel emitting ω . When a state is entered, the automaton outputs the associated symbol, that is, ω or χ .

Markov chains are weighted automata, where each transition has probability. The sum of all the outgoing probabilities of a state is equal to 1. When entering a state, the Markov chain outputs a symbol – called here an observation –

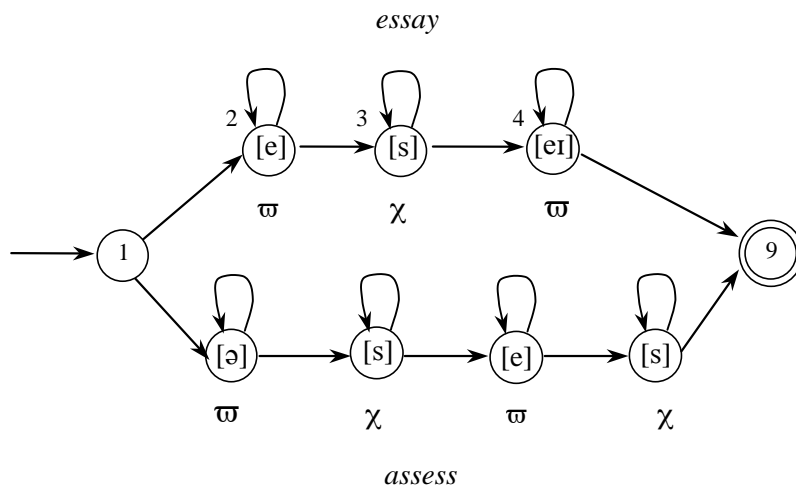


Fig. 1.11. An automaton representing the words *essay* [ˈesɪ] and *assess* [əˈses].

as for classical automata, and it associates a probability to strings of symbols it produces. Figure 1.12 shows an annotation of edges with numbers. According to Fig. 1.12, the probability of having sequence $\omega\omega\omega\omega\omega\omega\chi\chi\chi\chi\chi\omega\omega\omega\omega\omega$ is $0.5 \times 0.75^6 \times 0.25 \times 0.7^5 \times 0.3 \times 0.65^6 \times 0.35$.

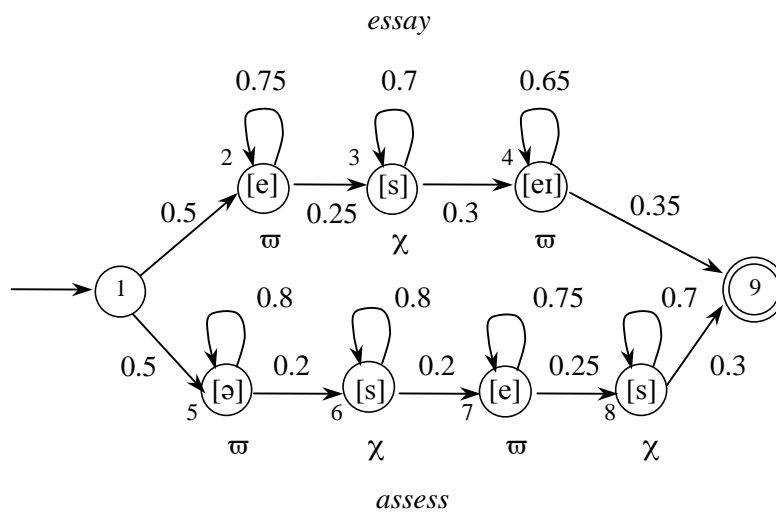


Fig. 1.12. A Markov chain with output probabilities.

1.4.6 Markov Chains in Prolog

Markov chains in Prolog are just an extension of simple automata. Let us describe the transitions of the chain in Fig. 1.12 using facts, where ω is transcribed as o, and χ as k:

```
% The start state
start(q1).

% The final state
final(q9).

% The transitions
transition(q1, o, q2, 0.5).
transition(q2, o, q2, 0.75).
transition(q2, k, q3, 0.25).
transition(q3, k, q3, 0.7).
transition(q3, o, q4, 0.3).
transition(q4, o, q4, 0.65).

transition(q1, o, q5, 0.5).
transition(q5, o, q5, 0.8).
transition(q5, k, q6, 0.2).
transition(q6, k, q6, 0.8).
transition(q6, o, q7, 0.2).
transition(q7, o, q7, 0.75).
transition(q7, k, q8, 0.25).
transition(q8, k, q8, 0.7).

silent(q4, q9, 0.35).
silent(q8, q9, 0.3).
```

The Prolog rules compute the chain probability by multiplying the transition probabilities:

```
accept(Observations, Probability) :-
    start(StartState),
    accept(Observations, StartState, 1.0, Probability).

accept([], State, Probability, Probability) :-
    final(State).

accept([Observation | Observations], State, Prob, ProbOut) :-
    transition(State, Observation, NextState, ProbTrans),
    NextProb is Prob * ProbTrans,
    accept(Observations, NextState, NextProb, ProbOut).

accept(Observations, State, ProbIn, ProbOut) :-
```

```

silent(State, NextState, ProbTrans),
NextProb is ProbIn * ProbTrans,
accept(Observations, NextState, NextProb, ProbOut).

?- accept([o, o, o, o, o, o, o, k, k, k, k, k, k, o, o,
o, o, o, o, o], Prob).
Prob = 2.96099e-005

```

1.4.7 Hidden Markov Models

From the Markov chain, we can go to a hidden Markov model. We saw in Chap. ?? that a HMM is a Markov chain, except that a state is no longer linked to a single output symbol but with all possible acoustic symbols and each symbol has a probability of occurring. Of course, some symbols will be much more probable than others. Using HMM, we have now a model of emission of spurious symbols since a state corresponding to a specific phoneme can output any symbol.

Let us take the words *essay* [ˈeseɪ] and *assay* [əˈseɪ], and let us suppose that we have a set of three acoustic symbols. Two acoustic symbols are related to two vowels, ω_1 to [e] and [eɪ] and ω_2 to [ə], and the third one, χ , to a consonant [s]. Let us annotate states with a probability distribution over these three symbols (Fig. 1.13). The first vowels of both words are similar. There are occasions, especially in fluent speech, where speakers will mix them up. The Markov model states here that one symbol is more frequent than others when a user utters a specific vowel. When uttering the vowel [e] of [ˈeseɪ], the probability of generating ω_1 is 0.6, ω_2 is 0.3, and χ is 0.1.¹ This corresponds to entering the top leftmost state of the network. On the contrary, the leftmost bottom state has a probability of generating ω_1 that is 0.3, ω_2 that is 0.7, and χ that is 0.

Recognition with HMM corresponds to searching the most likely path to traverse the automaton. Let us suppose that a speaker utters a word yielding the acoustic symbol string $\omega_1\omega_2\omega_1\chi\chi\omega_1\omega_1$. The upper path, [1, 2, 2, 2, 3, 3, 4, 4, 8], yields the value:

$$0.5 \times 0.6 \times 0.75 \times 0.3 \times 0.75 \times 0.6 \times 0.25 \times 1 \times 0.7 \times 1 \times 0.3 \times 0.65 \times 0.65 \times 0.65 \times 0.35 = 1.533 \cdot 10^{-4},$$

where the consonant symbols are both generated by state [3]. The lower one, [1, 5, 5, 5, 6, 6, 7, 7, 8], yields the value:

$$0.5 \times 0.3 \times 0.8 \times 0.7 \times 0.8 \times 0.3 \times 0.2 \times 1 \times 0.8 \times 1 \times 0.2 \times 0.65 \times 0.7 \times 0.65 \times 0.3 = 5.238 \cdot 10^{-5},$$

and hence this automaton would recognize the string as being the word *essay*.

In fact, there are other possible paths. The subchain $\omega_1\omega_2\omega_1\chi\chi$ could also have been generated by path [1, 2, 2, 2, 2, 3]. The second χ symbol is necessarily generated by state [3] since it is a barrier for vowels – the probability of them being 0. Two competing paths can then reach state [3] and output the same string. However, the likelihood for [1, 2, 2, 2, 2, 3] is

¹ Probabilities are fictitious.

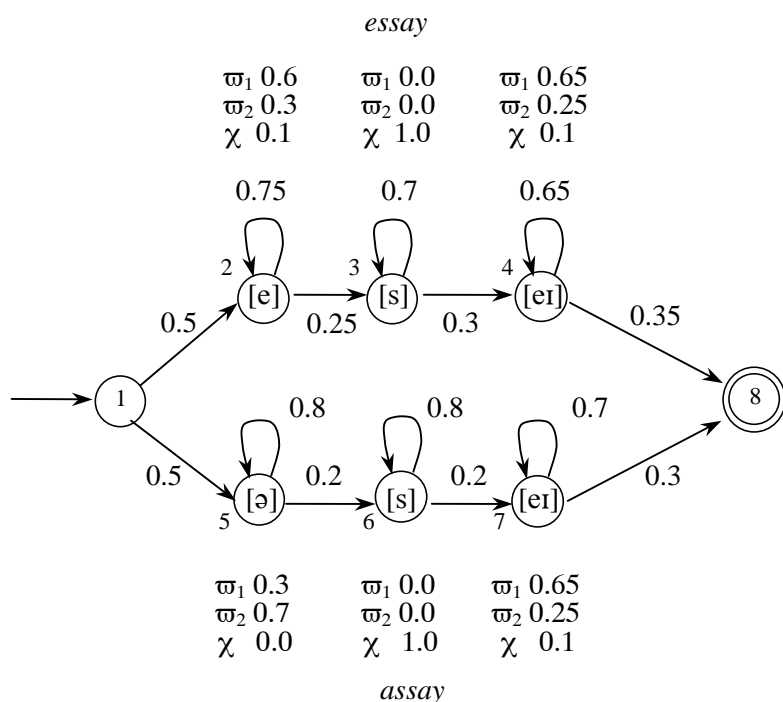


Fig. 1.13. A hidden Markov model.

$0.5 \times 0.6 \times 0.75 \times 0.3 \times 0.75 \times 0.6 \times 0.75 \times 0.1 \times 0.25 \times 1 = 5.7 \cdot 10^{-4}$,
 compared with a likelihood for path [1, 2, 2, 2, 3, 3] of

$$0.5 \times 0.6 \times 0.75 \times 0.3 \times 0.75 \times 0.6 \times 0.25 \times 1 \times 0.8 \times 1 = 6.1 \cdot 10^{-3},$$

when both consonant symbols are generated by state [3]. From then on, we are sure that successors of path [1, 2, 2, 2, 3] will be more likely than those of [1, 2, 2, 2, 2], and we can prune the former.

Such a pruning corresponds to the Viterbi optimization that we saw in Chap. ?? . It avoids searching all possible paths. When competing paths reach a state j having generated the same string of symbols, here $\omega_1\omega_2\omega_1\chi\chi$, the path with the highest probability is retained and others are discarded.

1.4.8 Hidden Markov Models in Prolog

The hidden Markov model is just an extension of the Markov chain program. We need to modify the facts to add the observation probabilities in Fig. 1.13. We transcribe ω_1 as o1, ω_2 as o2, and χ as k:

```
% The start state
start(q1).
```

```

% The final states
final(q8).

transition(q1, o1, q2, 0.5, 0.6).
transition(q1, o2, q2, 0.5, 0.3).
transition(q1, k, q2, 0.5, 0.1).
transition(q2, o1, q2, 0.75, 0.6).
transition(q2, o2, q2, 0.75, 0.3).
transition(q2, k, q2, 0.75, 0.1).
transition(q2, o1, q3, 0.25, 0.0).
transition(q2, o2, q3, 0.25, 0.0).
transition(q2, k, q3, 0.25, 1.0).
transition(q3, o1, q3, 0.7, 0.0).
transition(q3, o2, q3, 0.7, 0.0).
transition(q3, k, q3, 0.7, 1.0).
transition(q3, o1, q4, 0.3, 0.65).
transition(q3, o2, q4, 0.3, 0.25).
transition(q3, k, q4, 0.3, 0.1).
transition(q4, o1, q4, 0.65, 0.65).
transition(q4, o2, q4, 0.65, 0.25).
transition(q4, k, q4, 0.65, 0.1).
transition(q1, o1, q5, 0.5, 0.3).
transition(q1, o2, q5, 0.5, 0.7).
transition(q1, k, q5, 0.5, 0.0).
transition(q5, o1, q5, 0.8, 0.3).
transition(q5, o2, q5, 0.8, 0.7).
transition(q5, k, q5, 0.8, 0.0).
transition(q5, o1, q6, 0.2, 0.0).
transition(q5, o2, q6, 0.2, 0.0).
transition(q5, k, q6, 0.2, 1.0).
transition(q6, o1, q6, 0.8, 0.0).
transition(q6, o2, q6, 0.8, 0.0).
transition(q6, k, q6, 0.8, 1.0).
transition(q6, o1, q7, 0.2, 0.65).
transition(q6, o2, q7, 0.2, 0.25).
transition(q6, k, q7, 0.2, 0.1).
transition(q7, o1, q7, 0.7, 0.65).
transition(q7, o2, q7, 0.7, 0.25).
transition(q7, k, q7, 0.7, 0.1).

silent(q4, q8, 0.35).
silent(q7, q8, 0.3).

```

The Prolog rules compute the path probabilities by multiplying the transition probabilities by the observation probabilities:

```
accept(Observations, States, Prob) :-
    start(StartState),
    accept(Observations, StartState, 1.0, States, Prob).

accept([], State, Probability, [State], Probability) :-
    final(State).
accept([Observ | Observs], State, Prob, [State | States],
    ProbOut) :-
    transition(State, Observ, NextState, ProbTrans, ProbObserve),
    ProbObserve =\= 0.0,
    NextProb is Prob * ProbTrans * ProbObserve,
    accept(Observs, NextState, NextProb, States, ProbOut).
accept(Observs, State, ProbIn, [State | States], ProbOut) :-
    silent(State, NextState, ProbTrans),
    NextProb is ProbIn * ProbTrans,
    accept(Observs, NextState, NextProb, States, ProbOut).
```

We obtain the probability of one path using `accept/3`:

```
?- accept([o1, o2, o1, k, k, o1, o1], States, Prob).
```

```
States = [q1, q2, q2, q2, q2, q3, q4, q4, q8]
Prob = 1.64228e-005 ;
```

```
States = [q1, q2, q2, q2, q3, q3, q4, q4, q8]
Prob = 0.000153279 ;
```

```
States = [q1, q2, q2, q2, q3, q4, q4, q4, q8]
Prob = 1.42331e-005
```

Yes

Finally, we write the `most_probable_path/3` predicate to obtain the most probable path using the built-in predicates `findall/3` and `keysort/2`:

```
most_probable_path(Observ, MP_Path, MP_Prob) :-
    findall(-(Prob, States),
    accept(Observ, States, Prob), L),
    keysort(L, Sorted),
    append(_, [-(MP_Prob, MP_Path)], Sorted).
```

where `append(_, [-(MP_Prob, MP_Path)], Sorted)` finds the last element of the list `Sorted`.

```
?- most_probable_path([o1, o2, o1, k, k, o1, o1], Path, Prob).
Path = [q1, q2, q2, q2, q3, q3, q4, q4, q8]
Prob = 0.000153279
```

1.4.9 The Viterbi Algorithm in Prolog

The Viterbi algorithm optimizes the search by taking the path with the maximum probability among all the paths that lead to a state. We need to modify the program to consider the paths simultaneously. For sake of simplicity, we rewrite the silent transitions as ordinary transitions with the epsilon symbol:

```
transition(q4, e, q8, 0.35, 1.0).
transition(q7, e, q8, 0.3, 1.0).
```

The program proceeds in two steps. Given a set of paths and an observation, it computes all the possible paths that can generate the observation. It then discards the nonoptimal ones.

```
search(Observations, Paths) :-
    start(StartState),
    viterbi(Observations, [-(1.0, [StartState])], Paths).

viterbi([], Paths, OptimalPaths) :-
    findall(-(Prob, [State | Path]),
        (member(-(Prob, [State | Path]), Paths), final(State)),
        OptimalPaths).
viterbi([Observation | Observations], Paths, Result) :-
    extend_paths(Observation, Paths, NewPaths),
    keysort(NewPaths, SortedPaths),
    discard_paths(SortedPaths, OptimalPaths),
    viterbi(Observations, OptimalPaths, Result).
```

From a list of paths and an observation, the `extend_path/3` predicate extends all the paths in the list:

```
% extend_paths(+Observation, +Paths, -NewPaths)
extend_paths(_, [], []).
extend_paths(Observ, [Path | Paths], NewPaths) :-
    findall(NewPath, extend_path(Observ, Path, NewPath),
        FirstNewPaths),
    extend_paths(Observ, Paths, RestNewPaths),
    append(FirstNewPaths, RestNewPaths, NewPaths).

%extend_path(+Observation, +Path, -NewPath)
% Given an observation, extend_path/3 extends one path
extend_path(Observation, -(Prob, [State | Path]),
    -(NextProb, [NextState, State | Path])) :-
```

```

transition(State, Observation, NextState, ProbTrans,
           ProbObserve),
ProbObserve =\= 0.0,
NextProb is Prob * ProbTrans * ProbObserve.

```

The `discard_paths/2` predicate discards the non-optimal paths:

```

% discard_paths(+SortedKeyList, -OptimalList)

discard_paths([], []).
% There is another path leading to a same state with a
% higher probability. We discard it.
discard_paths([-(_, [State | _]) | SortedPaths],
              OptimalPaths) :-
  member(-(_, [State | _]), SortedPaths),
  discard_paths(SortedPaths, OptimalPaths).
discard_paths([- (Prob, [State | Path]) | SortedPaths],
              [- (Prob, [State | Path]) | OptimalPaths]) :-
  \+ member(-(_, [State | _]), SortedPaths),
  discard_paths(SortedPaths, OptimalPaths).

```

And finally:

```

?- search([o1, o2, o1, k, k, o1, o1, e], Paths).

Paths = [0.000153279-[q8, q4, q4, q3, q3, q2, q2, q1]]

```

1.4.10 Modeling Phones with Hidden Markov Models

We have applied hidden Markov modeling to whole words. In this case, given a predetermined number of states, we need to build a specific model for each word of vocabulary. This method is not very flexible because it does not scale up nicely when a new word has to be added. A preferred approach is to use acoustic models of phones as basic units instead of complete words. In this case, recognition uses a pronunciation modeling of words based on phonemes. A word acoustic model is obtained in two steps:

- Each word is represented by one or more phoneme strings corresponding to its possible pronunciations.
- Each phoneme of the word is mapped onto its Markov acoustic model.

The acoustic model of a word results then in the concatenation of elementary phoneme models taken from a finite set and specific to each language. This approach makes the introduction of a new word into the recognition vocabulary easier since it requires only its phonetic decomposition.

On most systems, the automata do not use isolated phone models, but instead use the triphone concept: a phone influenced by its left and right neighbors. The

phoneme /s/ in *essay* [ˈesɪ] is then modeled as [e] [s] [e] and is different from /s/ in *assay* [əˈseɪ], modeled as [ə] [s] [e]. Such phonemes are said to be phonemes in context. Phonemes in context increase considerably the number of modeling units because the number of units expands from 50 phones approximately to 50^3 , that is, 125,000. Fortunately, many similar triphones can be clustered together, which results in a reduction of this number.

The automaton model of a phoneme in context has three states corresponding to the left context influence, the middle, and the right context influence (Fig. 1.14).

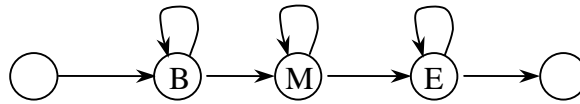


Fig. 1.14. HMM of a phone in context.

Most speech recognition systems, as the pioneering SPHINX system, use a more sophisticated automaton (Fig. 1.15) and feature a specific modeling for poorly articulated “function” words: “a”, “the”, “of”, ... (Lee et al., 1990). Looping transitions model slow pronunciation, and skipping transitions model faster pronunciations.

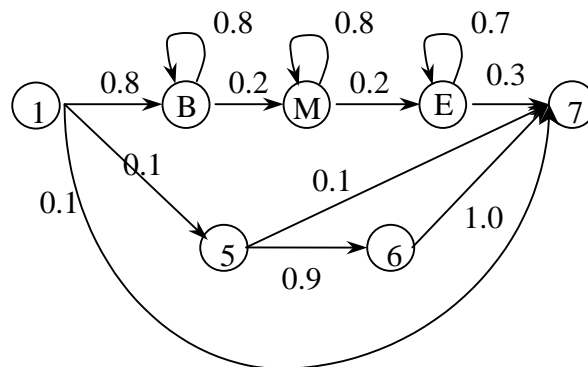


Fig. 1.15. The HMM phone model in the SPHINX system. After Lee et al. (1990).

The coefficients used in HMMs are determined from speech corpora annotated with their phonetic transcription. The coefficients are trained using the forward-backward algorithm seen in Chap. ?? . Corpora usually try to include a wide variety of speech accents so that models are valid for any kind of speaker.

1.4.11 Word Decoding

While Markov models deliver a probabilistic mapping of a string of acoustic symbols a_1, a_2, \dots, a_m onto a string of phonemes $\varphi_1, \varphi_2, \dots, \varphi_m$, the language model

$P(W) = P(w_1, w_2, \dots, w_n)$ applies a second probability to a word sequence. The complete speech recognition then consists in decoding word sequences w_1, w_2, \dots, w_n from phonemic strings and weighting them using the language model. This can be restated as a hypothesis search problem that matches words to contiguous subsequences of the string (Jelinek, 1997).

$$\begin{array}{rcccc}
 \text{words} & & \overbrace{\phantom{a_1^1, a_1^2, \dots, a_1^{m1}}} & \overbrace{\phantom{a_2^1, a_2^2, \dots, a_2^{m2}}} & \overbrace{\phantom{a_j^1, \dots, a_j^{mj}}} & \overbrace{\phantom{a_n^1, \dots, a_n^{mn}}} \\
 \text{phonemes} & \varphi_1^1, \varphi_1^2, \dots, \varphi_1^{m1} & \varphi_2^1, \varphi_2^2, \dots, \varphi_2^{m2} & \varphi_j^1, \dots, \varphi_j^{mj} & \varphi_n^1, \dots, \varphi_n^{mn} \\
 \text{ac.symbols} & a_1^1, a_1^2, \dots, a_1^{m1} & a_2^1, a_2^2, \dots, a_2^{m2}, \dots & a_j^1, \dots, a_j^{mj}, \dots & a_n^1, \dots, a_n^{mn}
 \end{array}$$

If the vocabulary contains k words v_1, v_2, \dots, v_k , w_1 is to be selected among k possibilities, w_2 among k possible choices again, and so on. Figure 1.16 shows a search tree corresponding to an utterance of a three-word sequence with a vocabulary of three words. The nodes of the tree represent HMMs mapping a word phonetic transcription and weighted by the language model,

$$P(W) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2).$$

For such a small vocabulary, there are already 27 paths to search. With a real-size vocabulary, it would be impossible to explore all the branches, and most systems use specific techniques to reduce the search space. Decoding is usually done in several stages: first synchronously with the acoustic modeling stage, and then in a second pass to refine results. The main techniques are based on the Viterbi search and the *fast match* algorithm derived from the A* search strategy.

The Viterbi search applies to HMMs of phonemes weighted by the language model. The Viterbi search enables us to optimize the number of competing paths. However, potential paths are still usually too numerous. To keep the search tractable, improbable paths are pruned according to a probability criterion. This is done while processing the acoustic symbol string. The paths reaching a state corresponding to the end of a word with a probability under a specific threshold are discarded. This threshold depends on the maximum probability that is estimated for all remaining ongoing paths while processing the speech frames.

The Viterbi search and pruning is also called a beam search. Since paths are pruned before completion, the algorithm does not guarantee to keep the most likely word sequence. However, it is possible to prune 80–90% of hypotheses without loss. The beam search results in an N -best list: the list of the N best utterances. Many continuous speech engines can complement decoding with a grammar made of phrase-structure rules that can constrain the search even more.

The complete description of word decoding is outside the scope of this book. Jelinek (1997) is a good reference on the topic, where he reviews search strategies in detail.

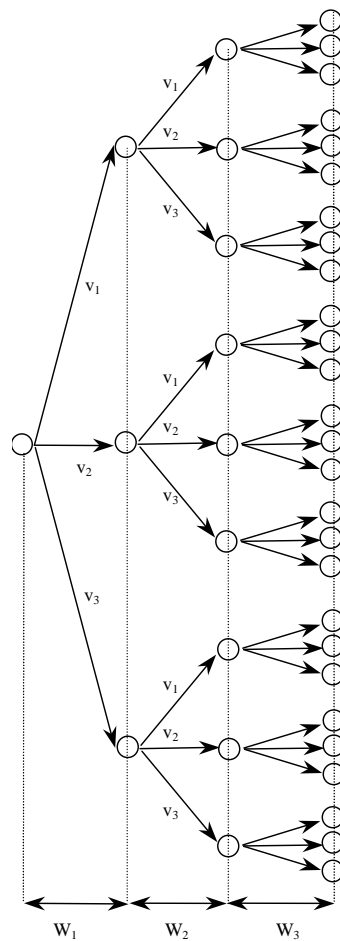


Fig. 1.16. Searching with a vocabulary of three words.

1.5 Application Programming Interfaces for Speech Processing

To develop a spoken system, language application programmers will likely resort to speech engines available on the market, unless they are bold enough to start a speech recognition system from scratch. Speech processing – recognition and synthesis – comes then as a packaged module that a developer can integrate into a larger application.

Most speech engines offer an interface to program them: a set of built-in functions that programmers can call from their application. Such function sets are called **application programming interfaces** (APIs). The developer of a conversational agent incorporates calls to the API functions in her/his program to control the speech engine and to get speech transcripts of commands, questions, etc. (Fig. 1.17). The ap-

plication programmer then has no need to know precisely what is inside the speech processing box; s/he has just to use relevant functions to start the engine, obtain the stream of recognized words, modify parameters, and so on.

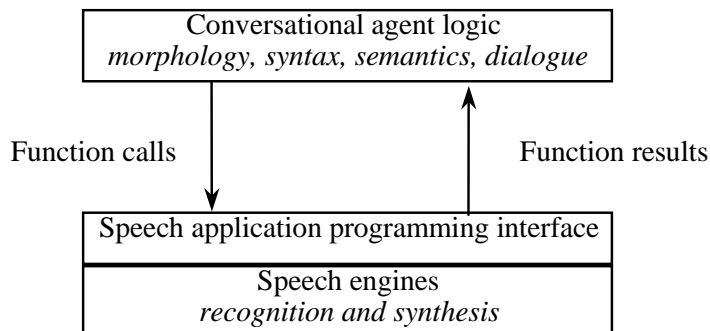


Fig. 1.17. Speech API and the rest of a conversational agent.

Speech APIs interact with the operating system they are running with, that is, MacOS, Unix, or MS Windows, and adopt their programming style. Common programming languages of these APIs include C, C++, Basic, or Java. There is unfortunately no instance of speech API in Prolog. Therefore, a Prolog program intended to process speech will have to call a “foreign” program written, for example, in C++ or in Java.

Common speech APIs generally offer two ways to program them. One is said to be the command and control mode, and the other the dictation mode. The dictation mode refers to freely dictated text. The command and control mode accepts only a limited set of words that are comparable to commands of a window interface, such as *File* and then *Open*, or *Save* or fixed templates, such as *Send mail to <Name>*, with *<Name>* being selected from a directory.

In the dictation mode, a program’s basic structure consists in initializing a speech session, loading the vocabularies and language models, setting up parameters, and finally processing speech input. In the command and control mode, a speech program also has to load an application grammar. The grammars required by speech engines are context-free and resemble DCG rules without variables, as this kiosk example from IBM ViaVoice (2001):

```

<kiosk> = <greeting1>? <greeting2>? <sentence1>
        | <greeting1>? <sentence2> .
<greeting1> = hello | excuse me | excuse me but .
<greeting2> = can you tell me | I need to know
        | please tell me .
<sentence1> = where <destination1> is located
        | where is <destination1>
        | where am I
  
```

```

| when will <transportation> <destination2>? arrive
| when <transportation> <destination2>? will arrive
| what time it is
| the local time
| the phone number of <destination1>
| the cost of <transportation> <destination2>? .
<sentence2> = I am lost
| I need help
| please help me
| help
| help me
| help me please .
<destination1> = a restaurant
| the <RestaurantType> restaurant
| <BusinessType>? <BusinessName> .
<RestaurantType> = best | nearest | cheapest | fastest .
<BusinessType> = a | the nearest .
<BusinessName> = filling station
| public rest room
| police station .
<transportation> = the <TransportType>? <TransportName> .
<TransportType> = next | first | last .
<TransportName> = bus | train .
<destination2> = to metro central
| to union station
| to downtown
| to national airport .

```

where “?” denotes an optional symbol, and “|” a disjunction.

The command and control mode obviously constrains the user more than the dictation mode: its grammar needs exact utterances that match a predefined set of templates, otherwise the user will not be recognized. Unlike command and control, the dictation mode has no grammar and is usually able to process much a larger vocabulary. The engine only uses the built-in language model.

The main advantage of command and control lies in its robustness to noise or to possible hesitations such as the many *ugh* embedded in natural speech. In this mode, the speech engine will attempt to match the utterance to the grammar rules and skip mumbles. The dictation mode, which does not have this sort of constraints, would probably result in more transcription errors. Dictation, however, is less prone to hesitations than spontaneous utterances, and therefore the dictation mode of speech engines can operate without a grammar.

1.6 Further Reading

A frequently cited reference on phonetics in English is that of Ladefoged (2001). Although not the most recent, a clear introduction to phonetics in French is due to Malmberg (1954). Pompino-Marschall (2003) is an introduction in German. A good guide to phonetic symbols is due to Pullum and Ladusaw (1996).

Spoken language processing by Huang et al. (2001) is an outstanding book that covers the field of speech recognition as well as speech synthesis. Dutoit (1997) provides an interesting introduction to speech synthesis. Grapheme-to-phoneme rules can be implemented by the way of finite-state automata. Roche and Schabes (1997) is a good survey of these techniques applied to language processing and phonetics. Among research topics in synthesis, prosody is a matter of active study. Delattre (1965) is an old but valuable reference where he describes and compares phonetic features in European languages.

Books on signal analysis are numerous. Introductory references include Lyons (2004), Bellanger (2006), and Kammeyer and Kroschel (2002). More specifically, Deller Jr. et al. (2000) is an excellent reference dedicated to speech signals. The book addresses many aspects of speech processing. The authors present and discuss theoretical background, methods and efficient computer implementations. Boite et al. (1999) and Vary et al. (1998) are also good references in French and German on speech signal processing. Jelinek (1997) is an outstanding presentation of statistical methods in language modeling and speech recognition. It is still worth reading Lee et al.'s (1990) description of the SPHINX system, which was one of the first continuous speech recognition systems. Teams from the University of Cambridge (Woodland et al., 1999) and LIMSI (Gauvain et al., 2000) are regular winners of the DARPA speech recognition competitions.

Speech engines for recognition as well as for synthesis are available from several vendors. Among APIs, IBM SAPI is probably the best designed. IBM's developer's guide (IBM, 2001) should enable a programmer to develop speech applications in C++. Other APIs include Microsoft's SASDK (SAS, 2005) and Sun's JSAPI (JSAPI, 1998).

Two notable systems mostly used for English are available with their source code from the Web. These are the Cambridge Hidden Markov Toolkit (Cambridge HTK, <http://htk.eng.cam.ac.uk/>) from the University of Cambridge for speech recognition and Festival from the University of Edinburgh for speech synthesis (<http://www.cstr.ed.ac.uk/projects/festival/>).

Exercises

1.1. Write ten grapheme-to-phoneme rules in a language you know. Take a short newspaper article and apply the rules manually using the longest-match algorithm. Measure the error rate.

1.2. Write a Prolog accepting grapheme-to-phoneme rules and using the longest-match algorithm. Apply the program on a short text with rules you have written.

- 1.3.** Take five sentences in a language that you know and, using four levels of pitch – 1, 2, 3, 4 – from low to high, try to annotate syllables of the sentences.
- 1.4.** Write a search algorithm in Prolog decoding words from a sequence of phonemes. Use a brute-force search strategy.
- 1.5.** Write a search algorithm decoding words from a sequence of phonemes. Incorporate a trigram language model and use a beam search strategy.
- 1.6.** Using a speech API, program an interactive dialogue system based upon the kiosk grammar.

Index

- allophone, 3
- Barney, H. L., 5
- Bellanger, M., 38
- Boite, R., 38
- Boula de Mareuil, P., 19
- broad transcription, 6
- cepstrum, 23
- Charpentier, F., 14
- Chomsky, N., 7, 16
- coarticulation, 6
- consonants, 6
- Delattre, P., 9, 10, 38
- Deller, J. R., 5, 38
- Divay, M., 16
- Dutoit, T., 38
- formant, 4
- Gauvain, J.-L., 38
- Hallahan, W. I., 16
- Halle, M., 7, 16
- hidden Markov model, 24
- Huang, X., 38
- IBM ViaVoice, 36
- Jelinek, F., 34, 38
- Kammeyer, K. D., 38
- Kroschel, K., 38
- Ladefoged, P., 38
- Ladusaw, W. A., 38
- Le Meur, P.-Y., 14
- Lee, K. F., 33, 38
- letter-to-phoneme rule, 15
- linear prediction, 22
- loudness, 8
- Lyons, R. G., 38
- Malmberg, B., 38
- Markov chain, 24
- Moulines, E., 14
- narrow transcription, 6
- Peterson, G. E., 5
- phone, 3
- phoneme, 3
- phoneme concatenation, 13
- phonetics, 2
- phonological rule, 7
- Pierrehumbert, J. B., 9
- pitch, 4, 8
- Pompino-Marschall, B., 38
- prosody, 8
- prosody generation, 18
- Pullum, G. K., 38
- quantity, 8
- Roche, E., 17, 38
- Schabes, Y., 17, 38
- sound capture, 2
- speech parameter, 22

- speech processing, application programming interfaces for, 35
- speech recognition, 1, 19
- speech recognition system, structure of, 20
- speech synthesis, 1, 12
- SPHINX, 33
- suprasegmental feature, 8
- text-to-speech converter, 13
- timbre, 5
- transcription ambiguity, 17
- TTS, 13
- Vary, P., 38
- Vitale, A. J., 16
- voicing, 6
- vowels, 4
- Woodland, P., 38
- word decoding, 33
- Yarowsky, D., 18

References

- Bellanger, M. (2006). *Traitement numérique du signal : Théorie et pratique*. Dunod, Paris, 8e édition.
- Boite, R., Boulard, H., Dutoit, T., Hancq, J., and Leich, H. (1999). *Traitement de la parole*. Presses Polytechniques et Universitaires Romandes, Lausanne.
- Boula de Mareüil, P., d'Alessandro, C., Beaugendre, F., and Lacheret-Dujour, A. (2001). Une grammaire en tronçons appliquée à la génération de la prosodie. *Traitement automatique des langues*, 42(1):115–143.
- Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row, New York.
- Delattre, P. (1965). *Comparing the Phonetic Features of English, French, German, and Spanish*. Julius Groos Verlag, Heidelberg.
- Delattre, P. (1966a). L'accent final en français: accent d'intensité, accent de hauteur, accent de durée. In *Studies in French and Comparative Phonetics*, pages 65–68. Mouton, The Hague.
- Delattre, P. (1966b). Les dix intonations de base du français. *The French Review*, 40(1):1–14.
- Deller Jr., J. R., Hansen, J. H. L., and Proakis, J. G. (2000). *Discrete-Time Processing of Speech Signals*. IEEE, New York.
- Divay, M. and Vitale, A. J. (1997). Algorithms for grapheme-phoneme translation for English and French: Applications for database searches and speech synthesis. *Computational Linguistics*, 23(4):495–523.
- Dutoit, T. (1997). *Introduction to Text-to-Speech Synthesis*. Kluwer Academic Publishers, Dordrecht.
- Gauvain, J.-L., Lamel, L., and Adda, G. (2000). The LIMSI 1999 hub-4e transcription system. In *Proceedings of the 2000 Speech Transcription Workshop*, University of Maryland. <http://www.nist.gov/speech/publications/tw00/index.htm>. Cited 28 October 2005.
- Hallahan, W. I. (1996). DECTalk software: Text-to-speech technology and implementation. *Digital Technical Journal*, 7(4):5–19.

- Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing. A Guide to Theory, Algorithm, and System Development*. Prentice Hall, Upper Saddle River.
- IBM (2001). *IBM ViaVoice SDK, SMAPI Developer's Guide, Version 7.0*. IBM.
- International Phonetic Association, editor (1999). *Handbook of the International Phonetic Association, A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press, Cambridge.
- Jelinek, F. (1997). *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, Massachusetts.
- JSAPI (1998). *Java Speech API Programmer's Guide*. Sun Microsystems.
- Kammeyer, K.-D. and Kroschel, K. (2002). *Digitale Signalverarbeitung. Filterung und Spektralanalyse mit MATLAB-Übungen*. Teubner, Stuttgart, 5. edition.
- Ladefoged, P. (2001). *A Course in Phonetics*. Harcourt College Publishers, Fort Worth, 4th edition.
- Le Meur, P.-Y. (1996). *Synthèse de parole par unités de taille variable*. Thèse de doctorat, ENST, Paris.
- Lee, K.-F., Hon, H.-W., and Reddy, R. (1990). An overview of the SPHINX speech recognition system. *IEEE Transactions on ASSP*, 38(1):35–45.
- Lyons, R. G. (2004). *Understanding Digital Signal Processing*. Addison-Wesley, Upper Saddle River, 2nd edition.
- Malmberg, B. (1954). *La phonétique*. Number 637 in *Que sais-je?* Presses universitaires de France, Paris.
- Moulines, E. and Charpentier, F. (1990). Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech Communication*, 9(5-6):453–467.
- Peterson, G. E. and Barney, H. L. (1952). Control methods used in a study of the vowels. *Journal of the Acoustical Society of America*, 24(2):175–184.
- Pierrehumbert, J. B. (1980). *The Phonology and Phonetics of English Intonation*. PhD thesis, Massachusetts Institute of Technology.
- Pompino-Marschall, B. (2003). *Einführung in die Phonetik*. De Gruyter, Berlin, 2. edition.
- Pullum, G. K. and Ladusaw, W. A. (1996). *Phonetic Symbol Guide*. University of Chicago Press, Chicago, 2nd edition.
- Roche, E. and Schabes, Y., editors (1997). *Finite-State Language Processing*. MIT Press, Cambridge, Massachusetts.
- SAS (2005). *Microsoft Speech Application Software Development Kit 1.1*. Microsoft.
- Vary, P., Heute, U., and Hess, W. (1998). *Digitale Sprachsignalverarbeitung*. Teubner, Stuttgart.
- Woodland, P., Hain, T., Moore, G., Niesler, T., Povey, D., Tuerk, A., and Whittaker, E. (1999). The 1998 HTK broadcast news transcription system: Development and results. In *Proceedings of the DARPA Broadcast News Workshop*, Herndon, Virginia. <http://www.nist.gov/speech/publications/darpa99/index.htm>. Cited 28 October 2005.

- Yarowsky, D. (1996). Homograph disambiguation in speech synthesis. In van Santen, J., Sproat, R., Olive, J. P., and Hirschberg, J., editors, *Progress in Speech Synthesis*, pages 159–175, Berlin Heidelberg New York. Springer Verlag.