If we set the number of decision-tree clustered context-dependent phonemes to be large, there will be fewer choices of long speech segments that match. For instance, in a system with 2000 generalized triphones, the phonetic context of the last phoneme of a long segment and the context of the target phoneme may be clustered together, whereas in a 3000-generalized-triphone system, both contexts may not be clustered together, so that the long segment cannot be used. This would be one example where using a larger number of generalized triphones hurts speech naturalness because the database of speech segments is limited. This problem could have been avoided if we didn't have to match generalized triphones and instead allowed context substitutions, yet penalized them with a corresponding cost. In the framework of decision-tree clustered context-dependent phonemes, this cost can be computed as the increase in entropy when those contexts are merged, using the methods described in Chapter 9. The larger the increase in entropy, the larger the penalty is when doing that context substitution between the candidate segment and the target segment. This approach gives more flexibility in the number of speech segments to be considered. In this case, there is a nonzero unit coarticulation cost associated with replacing one phonetic context with another.

Speech segments that have low HMM probability can be discarded, as they are probably not representative enough for that unit. Moreover, we can eliminate outliers: those units that have parameters too far away from the mean. Eliminating pitch outliers helps if prosody modification is to be done, as modifying pitch by more than a factor of 2 typically yields a decrease of quality, and by keeping units with average pitch, this is less likely to occur. Eliminating duration or amplitude outliers may signal an incorrect segmentation or a bad transcription [13].

16.3.3. Unit Inventory Design

The minimal procedure to obtain an acoustic inventory for a concatenative speech synthesizer consists of simply recording a number of utterances from a single speaker and labeling them with the corresponding text.

Since recording is often done in several sessions, it is important to maintain the recording conditions constant to avoid spectral or amplitude discontinuities caused by changes in recording conditions. The same microphone, room, and sound card should be used throughout all sessions [49].

Not all donor speakers are created equal. The choice of donor speaker can have a significant effect in voice quality (up to 0.3 MOS points on a 5-MOS scale) [7, 51, 52].

We can obtain higher-quality concatenative synthesis if the text read by the target speaker is representative of the text to appear in our application. This way we will be able to use longer units, and few concatenations will be needed.

Then the waveforms have to be segmented into phonemes, which is generally done with a speech recognition system operating in forced-alignment mode. Phonetic transcription, including alternate pronunciations, is generated automatically from text by the phonetic analysis module of Chapter 14. A large part of the inventory preparation includes checking

Amazon/VB Assets
Exhibit 1012
Page 843

correspondence between the text and corresponding waveform. Possible transcription errors may be flagged by phonemes whose durations are too far away from the mean (outliers) [13, 24].

Once we have the segmented and labeled recordings, we can use them as our inventory, or create smaller inventories as subsets that trade off memory size with quality [21, 25]. A database with a large number of utterances is generally required to obtain high-quality synthesis. It is noteworthy to analyze whether we can reduce the size of our database while obtaining similar synthesis quality on a given set of utterances. To do this, we can measure the cost incurred when we use a subset of the units in the database to synthesize our training database. A greedy algorithm can be used that at each stage eliminates the speech unit that increases the total distortion the least, repeating the approach until the desired size is achieved. This is an iterative analysis-by-synthesis algorithm.

The above procedure can also be used to find the set of units that have lowest cost in synthesizing a given text. For efficiency, instead of a large training text, we could use representative information from such text corpus, like the word trigrams with their corresponding counts, as an approximation.

In concatenative systems, you need to store a large number of speech segments, which could be compressed using any of the speech coding techniques described in Chapter 7. Since many such coders encode a frame of speech based on the previous one, you need to store this context for every segment you want to encode if you are to use such systems.

16.4. PROSODIC MODIFICATION OF SPEECH

One problem of segment concatenation is that it doesn't generalize well to contexts not included in the training process, partly because prosodic variability is very large. There are techniques that allow us to modify the prosody of a unit to match the target prosody. These prosody-modification techniques degrade the quality of the synthetic speech, though the benefits are often greater than the distortion introduced by using them because of the added flexibility.

The objective of prosodic modification is to change the amplitude, duration, and pitch of a speech segment. Amplitude modification can be easily accomplished by direct multiplication, but duration and pitch changes are not so straightforward.

We first present OLA and SOLA, two algorithms to change the duration of a speech segment. Then we introduce PSOLA, a variant of the above that allows for pitch modification as well.

16.4.1. Synchronous Overlap and Add (SOLA)

Time-scale modification of speech is very useful, particularly voice compression, as it allows a user to listen to a voice mail or taped lecture in a fraction of the time taken by the original segment user to listen to information The overlap-and-add (OLA) technique [12]

Amazon/VB Assets
Exhibit 1012
Page 844

shown in Figure 16.8 shows the analysis and synthesis windows used in the time compression. Given a Hanning window of length $2N$ and a compression factor of f, the analysis windows are spaced fN. Each analysis window multiplies the analysis signal, and at synthesis time they are overlapped and added together. The synthesis windows are spaced N samples apart. The use of windows such as Hanning allows perfect reconstruction when f equals 1.

In Figure 16.8, some of the signal appearance has been lost; note particularly some irregular pitch periods. To solve this problem, the synchronous overlap-and-add (SOLA) [45] allows for a flexible positioning of the analysis window by searching the location of the analysis window i around fNi in such a way that the overlap region had maximum correlation. The SOLA algorithm produces high-quality time compression. A mathematical formulation of PSOLA, an extension of both OLA and SOLA, is presented in Section 16.4.2.

While typically compression algorithms operate at a uniform rate, they have also been used in a nonuniform rate to take into account human perception, so that rapid transitions are compressed only slightly, steady sounds are compressed more, and pauses are compressed the most. It's reported in [11], that while uniform time compression can achieve a factor of 2.5 at most without degradation in intelligibility, nonuniform compression allows up to an average compression factor of 4.

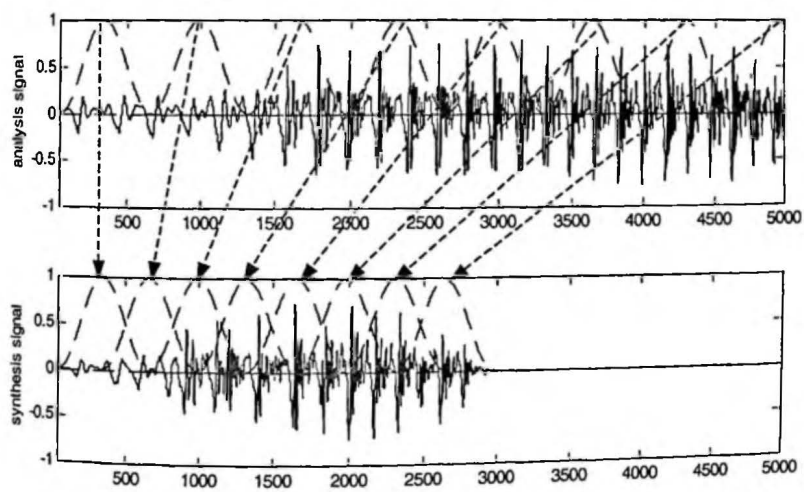


Figure 16.8 Overlap-and-add (OLA) method for time compression. Hanning windows of length $2N$, $N = 330$, are used to multiply the analysis signal, and resulting windowed signals are added. The analysis windows, spaced $2N$ samples, and the analysis signal $x[n]$ are shown on the top. The synthesis windows, spaced N samples apart, and the synthesis signal $y[n]$ are shown below. Time compression is uniform with a factor of 2. Pitch periodicity is somewhat lost, particularly around the fourth window.

Amazon/VB Assets
Exhibit 1012
Page 845

16.4.2. Pitch Synchronous Overlap and Add (PSOLA)

Both OLA and SOLA do duration modification but cannot do pitch modification. On the other hand, they operate without knowledge of the signal's pitch. The most widely used method to do pitch modification is called pitch synchronous overlap and add (PSOLA) [38, 39], though to do so it requires knowledge of the signal's pitch. This process is illustrated in Figure 16.9.

Let's assume that our input signal $x[n]$ is voiced, so that it can be expressed as a function of pitch cycles $x_i[n]$

$$x[n] = \sum_{i=-\infty}^{\infty} x_i[n - t_a[i]] \tag{16.6}$$

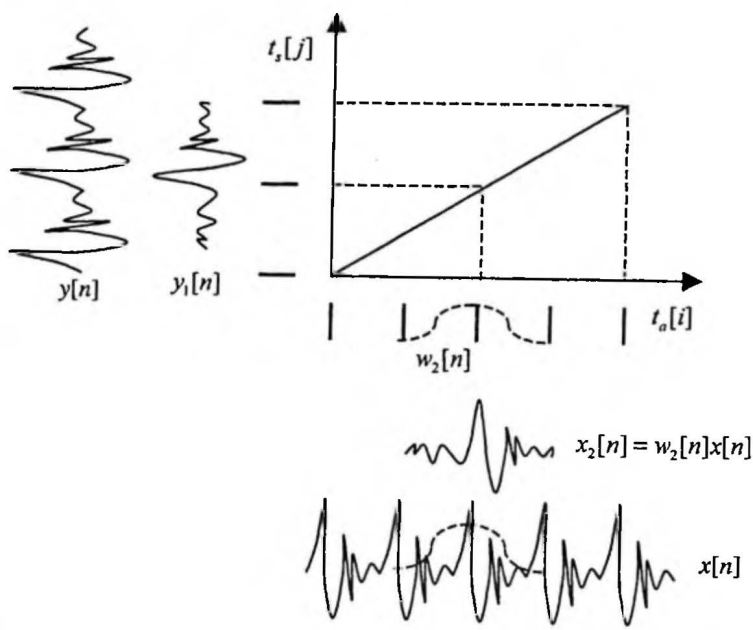


Figure 16.9 Mapping between five analysis epochs $t_a[i]$ and three synthesis epochs $t_s[j]$. Duration has been shortened by 40% and pitch period increased by 60%. Pitch cycle $x_2[n]$ is the product of the analysis window $w_2[n]$, in dotted line, with the analysis signal $x[n]$, which is aligned with analysis epochs $t_a[i]$. In this case, synthesis pitch cycle $y_1[n]$ equals $x_2[n]$ and also $y_0[n] = x_0[n]$ and $y_2[n] = x_3[n]$. Pitch is constant over time in this case.

Amazon/VB Assets
Exhibit 1012
Page 846

where $t_a[i]$ are the epochs of the signal, so that the difference between adjacent epochs $P_a[i] = t_a[i] - t_a[i-1]$ is the pitch period at time $t_a[i]$ in samples. The pitch cycle is a windowed version of the input

$$x_i[n] = w_i[n]x[n] \tag{16.7}$$

which requires the windows $w_i[n]$ to meet the following condition:

$$\sum_{i=-\infty}^{\infty} w_i[n - t_a[i]] = 1 \tag{16.8}$$

which can be accomplished with a Hanning window, or a trapezoidal window that spans two pitch periods.

Our goal is to synthesize a signal $y[n]$, which has the same spectral characteristics as $x[n]$ but with a different pitch and/or duration. To do this, we replace the analysis epoch sequence $t_a[i]$ with the synthesis epochs $t_s[j]$, and the analysis pitch cycles $x_i[n]$ with the synthesis pitch cycles $y_j[n]$:

$$y[n] = \sum_{j=-\infty}^{\infty} y_j[n - t_s[j]] \tag{16.9}$$

The synthesis epochs are computed so as to meet a specified duration and pitch contour, as shown in Figure 16.9. This is equivalent to an impulse train with variable spacing driving a time-varying filter $x_t[n]$ which is known for $t = t_a[i]$, as shown in Figure 16.10. The synthesis pitch cycle $y_j[n]$ is obtained via a mapping from the closest corresponding analysis pitch cycle $x_i[n]$. In the following sections we detail how to calculate the synthesis epochs and the synthesis pitch-cycle waveforms.

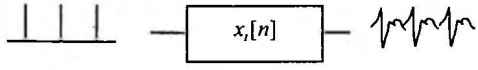


Figure 16.10 PSOLA technique as an impulse train driving a time-varying filter.

The term overlap-and-add derives from the fact that we use overlapping windows that we add together. The pitch-synchronous aspect comes from the fact that the windows are spaced a pitch period apart and are two pitch periods long. As you can see from Figure 16.9, the synthesis waveform has a larger pitch period than the analysis waveform and is shorter in duration.

For unvoiced speech, a set of epochs that are uniformly spaced works well in practice, as long as the spacing is smaller than 10 ms. If the segment needs to be stretched in such a way that these characteristic waveforms are repeated, an artificial periodicity would appear. To avoid this, the characteristic waveform that was to be repeated is flipped in time [38].

This approach is remarkably simple, yet it leads to high-quality prosody modification, as long as the voiced/unvoiced decision is correct and the epoch sequence is accurate.

Amazon/VB Assets
Exhibit 1012
Page 847

To do prosody modification, the PSOLA approach requires keeping the waveform of the speech segment and its corresponding set of epochs, or time marks. As you can see from Eq. (16.6), if no prosody modification is done, the original signal is recovered exactly.

16.4.3. Spectral Behavior of PSOLA

Let's analyze why this simple technique works and how. To do that let's consider the case of a speech signal $x[n]$ that is exactly periodic with period T_0 and can be created by passing an impulse train through a filter with impulse response $s[n]$:

$$x[n] = s[n] * \sum_{i=-\infty}^{\infty} \delta[n - iT_0] = \sum_{i=-\infty}^{\infty} s[n - iT_0] \tag{16.10}$$

If we know the impulse response $s[n]$, then we could change the pitch by changing T_0. The problem is how to estimate it from $x[n]$. Let's assume we want to build an estimate $\tilde{s}[n]$ by multiplying $x[n]$ by a window $w[n]$:

$$\tilde{s}[n] = w[n]x[n] \tag{16.11}$$

The Fourier transform of $x[n]$ in Eq. (16.10) is given by

$$X(\omega) = \frac{2\pi}{T_0} S(\omega) \sum_{k=0}^{T_0-1} \delta(\omega - k\omega_0) = \frac{2\pi}{T_0} \sum_{k=0}^{T_0-1} S(k\omega_0)\delta(\omega - k\omega_0) \tag{16.12}$$

where $\omega_0 = 2\pi / T_0$. The Fourier transform of $\tilde{s}[n]$ can be obtained using Eqs. (16.11) and (16.12):

$$\tilde{S}(\omega) = \frac{1}{2\pi} W(\omega) * X(\omega) = \sum_{k=0}^{T_0-1} S(k\omega_0) \frac{W(\omega - k\omega_0)}{T_0} \tag{16.13}$$

If the window $w[n]$ is pitch synchronous, a rectangular window with length T_0 or a Hanning window with length $2T_0$, for example, then the above estimate is exact at the harmonics, i.e., $\tilde{S}(k\omega_0) = S(k\omega_0)$. because the window leakage terms are zero at the harmonics. In-between harmonics, $\tilde{S}(\omega)$ is an interpolation using $W(\omega)$, the transfer function of the window. If we use a rectangular window, the values of $S(\omega)$ in between $S(k\omega_0)$ and $S((k+1)\omega_0)$ are not determined only by those two harmonics, because the leakage from the other harmonics is not negligible. The use of a Hanning window drastically attenuates this leakage, so the estimate of the spectral envelope is better. This is what PSOLA is doing: getting an estimate of the spectral envelope by using a pitch-synchronous window.

Since it is mathematically impossible to recover $S(\omega)$ for a periodic signal, it is reasonable to fill in the remaining values by interpolation with the main lobes of the transform of the window. This approach works particularly well if the harmonics form a dense sam-

Amazon/VB Assets
Exhibit 1012
Page 848

pling of the spectral envelope, which is the case for male speakers. For female speakers, where the harmonics may be spaced far apart, the spectral envelope estimated by interpolating across harmonics could be far different from the real envelope.

16.4.4. Synthesis Epoch Calculation

In practice, we want to generate a set of synthesis epochs $t_s[j]$ given a target pitch period $P_s(t)$. If the desired pitch period $P_s(t) = P$ is constant, then the synthesis epochs are given by $t_s[j] = jP$.

In general the desired pitch period $P_s(t)$ is a function of time. Intuitively, we could compute $t_s[j+1]$ in terms of the previous epoch $t_s[j]$ and the pitch period at that time:

$$t_s[j+1] - t_s[j] = P_s(t_s[j]) \tag{16.14}$$

though this is an approximation, which happens to work well if $P_s(t)$ changes slowly over time.

Now we derive an exact equation, which also can help us understand pitch-scale and time-scale modifications of the next few sections. Epoch $t_s[j+1]$ can be computed so that the distance between adjacent epochs $t_s[j+1] - t_s[j]$ equals the average pitch period in the region $t_s[j] \le t < t_s[j+1]$ between them (see Figure 16.11). This can be done by the following expression

$$t_s[j+1] - t_s[j] = \frac{1}{t_s[j+1] - t_s[j]} \int_{t_s[j]}^{t_s[j+1]} P_s(t)\,dt \tag{16.15}$$

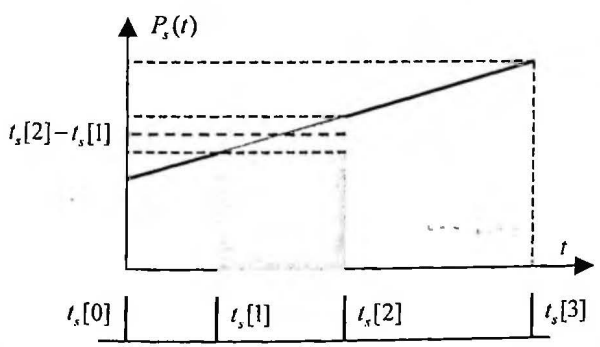


Figure 16.11 The desired pitch period $P_s(t)$ is a linearly increasing function of time such that the pitch period is doubled by the end of the segment. The four synthesis epochs $t_s[j]$ are computed to satisfy Eq. (16.15). In particular, $t_s[2]$ is computed such that $t_s[2] - t_s[1]$ equals the average pitch period in that region. Note that the growing spacing between epochs indicates that pitch is growing over time.

Amazon/VB Assets
Exhibit 1012
Page 849

It is useful to consider the case of $P_s(t)$ being linear with t in that interval:

$$P_s(t) = P_s\left(t_s[j]\right) + b\left(t - t_s[j]\right) \tag{16.16}$$

so that the integral in Eq. (16.15) is given by

$$\int_{t_s[j]}^{t_s[j+1]} P_s(t)dt = \delta_j \left[P(t_s[i]) + b\frac{\delta_j}{2} \right] \tag{16.17}$$

where we have defined δ_j as

$$\delta_j = t_s[j+1] - t_s[j] \tag{16.18}$$

Inserting Eqs. (16.17) and (16.18) into Eq. (16.15), we obtain

$$\delta_j = P(t_s[i]) + b\frac{\delta_j}{2} \tag{16.19}$$

which, using Eq. (16.18), gives a solution for epoch $t_s[j+1]$ as

$$t_s[j+1] - t_s[j] = \delta_j = \frac{P_s(t_s[j])}{(1 - b/2)} \tag{16.20}$$

from the previous epoch $t_s[j]$, the target pitch at that epoch $P_s(t_s[j])$, and the slope b. We see that Eq. (16.14) is a good approximation to Eq. (16.20) if the slope b is small.

Evaluating Eq. (16.16) for $t_s[j+1]$ results in an expression for $P_s(t_s[j+1])$

$$P_s(t_s[j+1]) = P_s\left(t_s[j]\right) + b\left(t_s[j+1] - t_s[j]\right) \tag{16.21}$$

Equations (16.20) and (16.21) can be used iteratively. It is important to note that Eq. (16.20) requires $b < 2$ in order to obtain meaningful results, which fortunately is always the case in practice.

When synthesizing excitations for speech synthesis, it is convenient to specify the synthesis pitch period $P_s(t)$ as a piecewise linear function of time. In this case, Eq. (16.20) is still valid as long as $t_s[j+1]$ falls within the same linear segment. Otherwise, the integral in Eq. (16.17) has two components, and a second-order equation needs to be solved to obtain $t_s[j+1]$.

Amazon/VB Assets
Exhibit 1012
Page 850

16.4.5. Pitch-Scale Modification Epoch Calculation

Sometimes, instead of generating an epoch sequence given by a function $P_s(t)$, we want to modify the epoch sequence of an analysis signal with epochs $t_a[i]$ by changing its pitch while maintaining its duration intact. This is called *pitch-scale* modification. To obtain the corresponding synthesis epochs, let's assume that the pitch period $P_a(t)$ of the analysis waveform at time t is constant and equals the difference between both epochs

$$P_a(t) = t_a[i+1] - t_a[i] \tag{16.22}$$

as seen in Figure 16.12.

The pitch period of the synthesis waveform $P_s(t)$ at the same time t now falls in between epochs j and $j+1$

$$t_s[j] \le t < t_s[j+1] \tag{16.23}$$

with $t_s[j]$ being the time instant of the j epoch of the synthesis waveform. Now, let's define a relationship between analysis and synthesis pitch periods

$$P_s(t) = \beta(t) P_a(t) \tag{16.24}$$

where $\beta(t)$ reflects the pitch-scale modification factor, which, in general, is a function of time. Following the derivation in Section 16.4.4, we compute the synthesis epoch $t_s[j+1]$ so that

$$t_s[j+1] - t_s[j] = \frac{1}{t_s[j+1] - t_s[j]} \int_{t_s[j]}^{t_s[j+1]} \beta(t) P_a(t) dt \tag{16.25}$$

which reflects the fact that the synthesis pitch period at time t is the average pitch period of the analysis waveform times the pitch-scale modification factor. Since $\beta(t)P_a(t)$ is piecewise linear, we can use the results of Section 16.4.4 to solve for $t_s[j+1]$. In general, it needs to be solved recursively, which results in a second-order equation if $\beta(t)$ is a constant or a linear function of t.

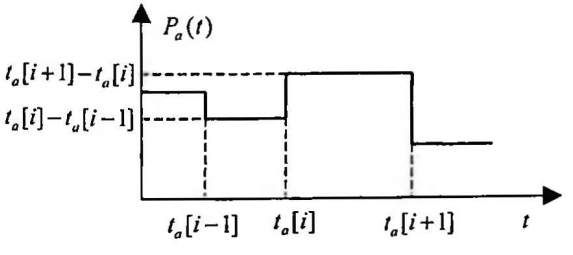


Figure 16.12 Pitch period of the analysis waveform as a function of time. It is a piecewise constant function of time.

Amazon/VB Assets
Exhibit 1012
Page 851

16.4.6. Time-Scale Modification Epoch Calculation

Time-scale modification of speech involves changing the duration of a speech segment while maintaining its pitch intact. This can be realized by defining a mapping $t_s = D(t_a)$, a time-warping function, between the original signal and the modified signal. It is useful to define the duration modification rate $\alpha(t)$ from which the mapping function can be derived:

$$D(t) = \int_0^t \alpha(\tau) d\tau \tag{16.26}$$

Let's now assume that the duration modification rate $\alpha(t) = \alpha$ is constant, so that the mapping $D(t)$ in Eq. (16.26) is linear. If $\alpha > 1$, we are slowing down the speech, whereas if $\alpha < 1$, we are speeding it up. Let's consider time t in between epochs i and $i + 1$ so that $t_a[i] \le t < t_a[i+1]$:

$$\begin{aligned} D(t_a[0]) &= 0 \\ D(t) &= D(t_a[i]) + \alpha(t - t_a[i]) \end{aligned} \tag{16.27}$$

So that the relationship between analysis and synthesis pitch periods is given by

$$P_s(D(t)) = P_a(t) \tag{16.28}$$

To solve this it is useful to define a stream of virtual time instants $t'_a[j]$ in the analysis signal related to the synthesis time instants by

$$t_s[j] = D(t'_a[j]) = \alpha t'_a[j] \tag{16.29}$$

as shown in Figure 16.13.

Now we try to determine $t_s[j+1]$ such that $t_s[j+1] - t_s[j]$ is equal to the average pitch period in the original time signal between $t'_a[j]$ and $t'_a[j+1]$:

$$t_s[j+1] - t_s[j] = \frac{1}{t'_a[j+1] - t'_a[j]} \int_{t'_a[j]}^{t'_a[j+1]} P_a(t) dt \tag{16.30}$$

which, using Eq. (16.29), results in

$$t_s[j+1] - t_s[j] = \frac{\alpha}{t_s[j+1] - t_s[j]} \int_{t_s[j]/\alpha}^{t_s[j+1]/\alpha} P_a(t) dt \tag{16.31}$$

which again results in a second-order equation if $P_a(t)$ is piecewise constant or linear in t.

Amazon/VB Assets
Exhibit 1012
Page 852

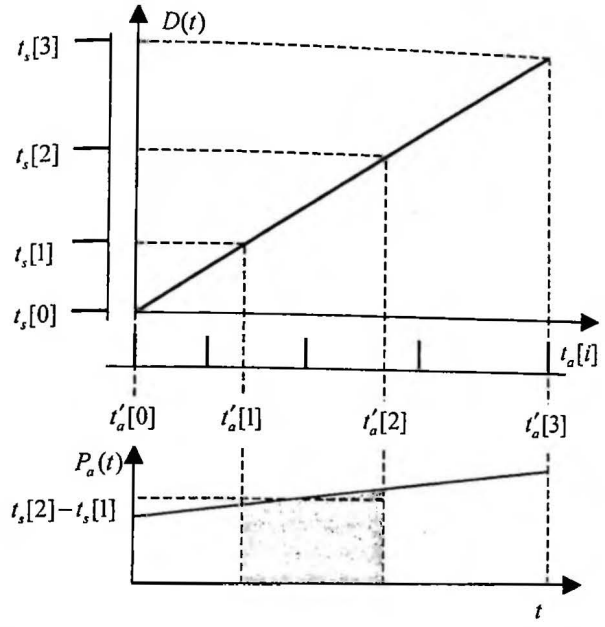


Figure 16.13 Time-scale modification of speech. The five analysis epochs $t_a[i]$ are shown in the x-axis and the four synthesis epochs $t_s[i]$ in the ordinate. Duration is shortened by 25% while maintaining the same pitch period. The corresponding virtual analysis epochs $t'_a[i]$ are obtained through the mapping $D(t)$, a linear transformation with $\alpha = 0.75$.

16.4.7. Pitch-Scale Time-Scale Epoch Calculation

The case of both pitch-scale and time-scale modification results in a combination of Eqs. (16.25) and (16.31):

$$t_s[j+1]-t_s[j]=\frac{\alpha}{t_s[j+1]-t_s[j]}\int_{t_s[j]/\alpha}^{t_s[j+1]/\alpha}\beta(t)P_a(t)dt \tag{16.32}$$

which again results in a second-order equation if $\beta(t)P_a(t)$ is piecewise constant or linear in t.

16.4.8. Waveform Mapping

The synthesis pitch waveforms can be computed through linear interpolation. Suppose that $t_a[i] \le t'_a[j] < t_a[i+1]$, then $y_j[n]$ is given by

Amazon/VB Assets
Exhibit 1012
Page 853

$$y_j[n] = (1-\gamma_j)x_i[n] + \gamma_j x_{i+1}[n] \tag{16.33}$$

where γ_j is given by

$$\gamma_j = \frac{t'_a[j] - t_a[i]}{t_a[i+1] - t_a[i]} \tag{16.34}$$

Using this interpolation for voiced sounds results in smooth speech. For unvoiced speech, this interpolation results in a decrease of the amount of aspiration. Since smoothness is not a problem in those cases, the interpolation formula above is not used for unvoiced frames. A simplification of this linear interpolation consists of rounding γ_j to 0 or 1 and, thus, selecting the closest frame.

16.4.9. Epoch Detection

In the PSOLA approach, the analysis epochs $t_a[i]$ were assumed known. In practice this is not the case and we need to estimate them from the speech signal. There can be errors if the pitch period is not correctly estimated, which results in a rough, noisy voice quality. But estimating the epochs is not a trivial task, and this is the most sensitive part of achieving prosody modification in PSOLA.

Most pitch trackers attempt to determine F0 and not the epochs. From the $t_a[i]$ sequence it is easy to determine the pitch, since $P(t) = t_a[i+1] - t_a[i]$ for $t_a[i] < t < t_a[i+1]$. But from the pitch $P(t)$ the epoch placement is not uniquely determined, since the time origin is unspecified.

Common pitch tracking errors, such as pitch doubling, pitch halving, or errors in voiced/unvoiced decisions, result in rough speech. While manual pitch marking can result in accurate pitch marks, it is time consuming and error prone as well, so automatic methods have received a great deal of attention.

A method that attains very high accuracy has been proposed through the use of an *electroglottograph* (EGG) [32]. It consists of a pair of electrodes strapped around the neck at both sides of the larynx that measures the impedance of the larynx. Such a device, also called *laryngograph*, delivers a periodic signal when the vocal cords are vibrating and no signal otherwise. The pitch shape of a laryngograph signal is fairly stationary, which makes it relatively easy to determine the epochs from it (see Figure 16.14).

High-quality epoch extraction can be achieved by performing peak picking on the derivative of the laryngograph signal. Often, the derivative operation is accomplished by a first-order preemphasis filter $H[z] = 1 - \alpha z^{-1}$, with α being close to 1 (0.95 is a good choice).

In practice, the signal is preprocessed to filter out the low frequencies (lower than 100 Hz) and high frequencies (higher than 4 kHz). This can be done with rectangular window filters that are quite efficient and easy to implement. There is a significant amount of energy outside this band that does not contribute to epoch detection, yet it can complicate the process, as can be seen in Figure 16.14, so this bandpass filtering is quite important.

Amazon/VB Assets
Exhibit 1012
Page 854

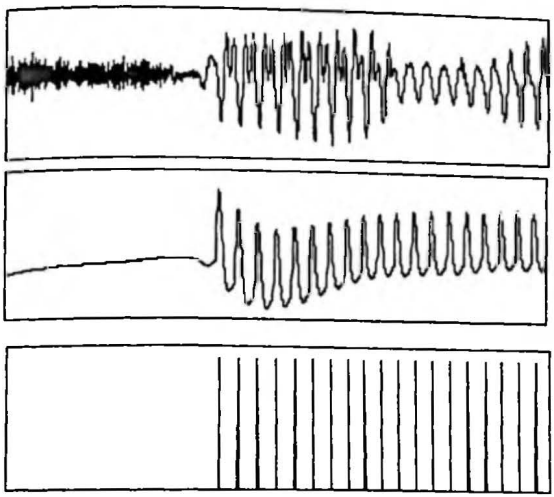


Figure 16.14 Speech signal, laryngograph signal, and its corresponding epochs.

The preemphasized signal exhibits peaks that are found by thresholding. The quality of this epoch detector has been evaluated on recordings from two female and four male speakers, and the voiced/unvoiced decision errors are lower than 1%. This is definitely acceptable for our prosody-modification algorithms. The quality of prosody modification with the epochs computed by this method can vastly exceed the quality achieved when standard pitch trackers (as described in Chapter 6) are used on the original speech signal [2].

16.4.10. Problems with PSOLA

The PSOLA approach is very effective in changing the pitch and duration of a speech segment if the epochs are determined accurately. Even assuming there are no pitch tracking errors, there can be problems when concatenating different segments:

- *Phase mismatches.* Even if the pitch period is accurately estimated, mismatches in the positioning of the epochs in the analysis signal can cause glitches in the output, as can be seen in Figure 16.15. The MBROLA [15] technique, an attempt to overcome phase mismatches, uses the time-domain PSOLA method for prosody modification, but the pitch cycles have been preprocessed so that they have a fixed phase. The advantage is that the spectral smoothing can be done by directly interpolating the pitch cycles in the time domain without adding any extra complexity. Since MBROLA sets the phase to a constant, the algorithm is more robust to phase errors in the epoch detection. Unfortunately, setting the phases constant incurs the added perceived noise described before.

Amazon/VB Assets
Exhibit 1012
Page 855

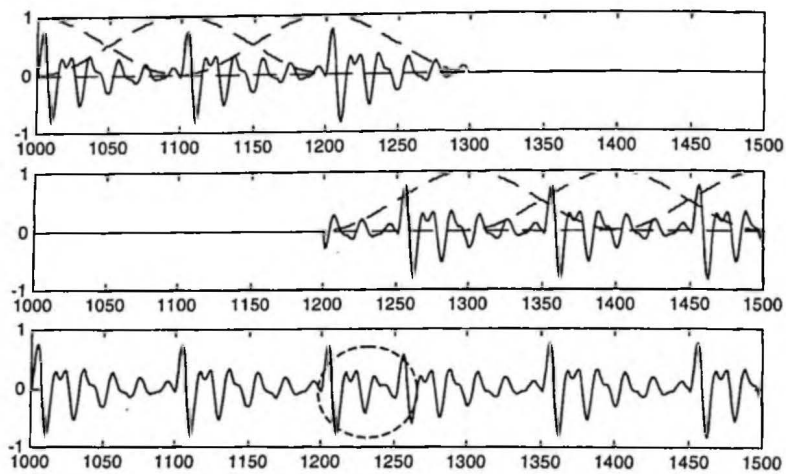


Figure 16.15 Phase mismatches in unit concatenation. Waveforms are identical, but windows are not centered on the same relative positions within periods.

- *Pitch mismatches.* These occur even if there are no pitch or phase errors during the analysis phase. As shown in Section 16.4.3, if two speech segments have the same spectral envelope but different pitch, the estimated spectral envelopes are not the same, and, thus, a discontinuity occurs (see Figure 16.16). In addition, pitch and timbre are not independent. Even when producing the same sound in the same phonetic context, a vastly different pitch will likely result in a different spectral envelope. This effect is particularly accentuated in the case of opera singers, who move their formants around somewhat so that the harmonics fall near the formant values and thus produce higher output.
- *Amplitude mismatch.* A mismatch in amplitude across different units can be corrected with an appropriate amplification, but it is not straightforward to compute such a factor. More importantly, the timbre of the sound will likely change with different levels of loudness.
- *Buzzy voiced fricatives.* The PSOLA approach doesn't handle well voiced fricatives that are stretched considerably because of added buzziness (repeating frames induces periodicity at the high frequency that wasn't present in the original signal) or attenuation of the aspirated component (if frames are interpolated).

Amazon/VB Assets
Exhibit 1012
Page 856

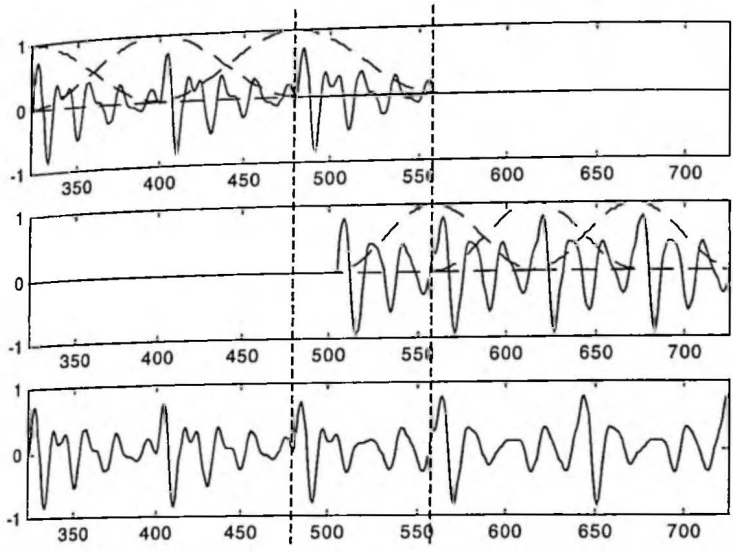


Figure 16.16 Pitch mismatches in unit concatenation. Two synthetic vowels were generated with a pitch of 138 Hz (top) and 197 Hz (middle) and exactly the same transfer function. There is no pitch tracking error, and windows are positioned coherently (no phase mismatch). The pitch of the second wave is changed through PSOLA to match the pitch of the first wave. There is a discontinuity in the resulting waveform and its spectrum (see Section 16.4.3), which is an artifact of the way the PSOLA approach estimates the spectral envelope.

16.5. SOURCE-FILTER MODELS FOR PROSODY MODIFICATION

The largest problem in concatenative synthesis occurs because of spectral discontinuities at unit boundaries. The methods described in Section 16.3 significantly reduce this problem but do not eliminate it. While PSOLA can do high-quality prosody modification on speech segments, it doesn't address these spectral discontinuities occurring at unit boundaries. It would be useful to come up with a technique that allows us to smooth these spectral discontinuities. In addition, PSOLA introduces buzziness for overstretched voiced fricatives. In the following sections we describe a number of techniques that have been proposed to cope with these problems and that are based on source-filter models.

Amazon/VB Assets
Exhibit 1012
Page 857

The use of source-filter models allow us to modify the source and filter separately and thus maintain more control over the resulting synthesized signal. In Section 16.5.1 we study an extension of PSOLA that allows filter modification as well for smoothing purposes. Section 16.5.2 describes mixed excitation models that also allow for improved voiced fricatives. Finally, Section 16.5.3 studies a number of voice effects that can be achieved with a source-filter model.

16.5.1. Prosody Modification of the LPC Residual

A method known as LP-PSOLA that has been proposed to allow smoothing in the spectral domain is to do PSOLA on LPC residual. This approach, thus, implicitly uses the LPC spectrum as the spectral envelope instead of the spectral envelope interpolated from the harmonics (see Section 16.4.3) when doing F0 modification. If the LPC spectrum is a better fit to the spectral envelope, this approach should reduce the spectral discontinuities due to different pitch values at the unit boundaries. LP-PSOLA reduces the *bandwidth widening*. In practice, however, this hasn't proven to offer a significant improvement in quality, possibly because the spectral discontinuities due to coarticulation dominate the overall quality.

The main advantage of this approach is that it allows us to smooth the LPC parameters around a unit boundary and thus obtain smoother speech. Since smoothing the LPC parameters directly may lead to unstable frames, other equivalent representations, such as line spectral frequencies, reflection coefficients, log-area ratios, or autocorrelation coefficients, are used instead. The use of a long window for smoothing may blur sharp spectral changes that occur in natural speech. In practice, a window of 20–50 ms centered around the boundary has been proven useful.

While time-domain PSOLA has low computational complexity, its use in a concatenative speech synthesizer generally requires a large acoustic inventory. In some applications this is unacceptable, and it needs to be compressed using any of the coding techniques described in Chapter 7. You need to keep in mind that to use such encoders you need to store the coder's memory so that the first frame of the unit can be accurately encoded. The combined decompression and prosody modification is not as computationally efficient as time-domain PSOLA alone, so that the LP-PSOLA approach may offer an effective tradeoff, given that many speech coders encode the LPC parameters anyway.

16.5.2. Mixed Excitation Models

The block diagram of PSOLA shown in Figure 16.10 for voiced sounds also works for unvoiced sounds by choosing arbitrary epochs. The time-varying filter of Figure 16.10 can be kept in its time-domain form or in the frequency domain $X_t[k]$ by taking the FFT of $x_t[n]$.

It has been empirically shown that for unvoiced frames, the phase of $X_t[k]$ is unimportant as long as it is random. Thus, we can pass white noise through a filter with magnitude response $|X_t[k]|$ and obtain perceptually indistinguishable results. This reduced representation is shown in Figure 16.17. Moreover, it has been shown that the magnitude spectrum

Amazon/VB Assets
Exhibit 1012
Page 858

does not need to be encoded accurately, because it doesn't affect the synthesized speech. The only potential problem with this model occurs when voiced frames are incorrectly classified as unvoiced.

Maintaining the phase of $X_t[k]$ is perceptually important for voiced sounds. If it is set to 0, two audible distortions appear: the reconstructed speech exhibits a noisy quality, and voiced fricatives sound buzzy.

The perceived noise may come from the fact that a listener who hears a formant, because of its amplitude spectrum, also expects the 180° phase shift associated with a complex pole. In fact, it is not the absolute phase, but the fact that if the formant frequency/bandwidth changes with time, there is a phase difference over time. If such a phase is not present, scene analysis done in the auditory system may match this to noise. This effect can be greatly attenuated if the phase of the residual in LP-PSOLA is set to 0, possibly because the LPC coefficients carry most of the needed phase information.

The buzziness in voiced fricatives is the result of setting phase coherence not only at low frequencies but also at high frequencies, where the aspiration component dominates. This is the result of treating the signal as voiced, when it has both a voiced and an unvoiced component. In fact, most voiced sounds contain some aperiodic component, particularly at high frequencies. The amount of aspiration present in a sound is called *breathiness*. Female speech tends to be more breathy than male speech [29]. Mixed-excitation models, such as those in Figure 16.18, are then proposed to more accurately represent speech.

Such a model is very similar to the waveform-interpolation coding approach of Chapter 7, and, hence, we can leverage much of what was described there regarding the estimation of $x_t'[n]$ and $x_t''[n]$. This approach allows us to integrate compression with prosody modification.

The harmonic-plus-noise [50] model decomposes the speech signal $s(t)$ as a sum of a random component $s_r(t)$ and a harmonic component $s_p(t)$

$$s(t) = s_r(t) + s_p(t) \tag{16.35}$$

where $s_p(t)$ uses the sinusoidal model described in Chapter 7:

$$s_p(t) = \sum_{k=1}^{K(t)} A_k(t)\cos(k\theta(t) + \phi_k(t)) \tag{16.36}$$

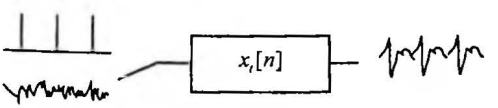


Figure 16.17 Speech synthesis model with white noise or an impulse train driving a time-varying filter.

Amazon/VB Assets
Exhibit 1012
Page 859

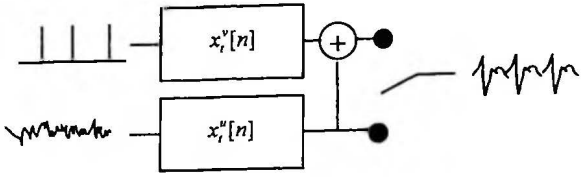


Figure 16.18 Mixed excitation model for speech synthesis.

where $A_k(t)$ and $\phi_k(t)$ are the amplitude and phase at time t of the kth harmonic, and $\theta(t)$ is given by

$$\theta(t) = \int_{-\infty}^{t} \omega_0(l)dl \tag{16.37}$$

16.5.3. Voice Effects

One advantage of using a spectral representation like those described in this section is that several voice effects can be achieved relatively easily, such as whisper, voice conversion, and echo/reverberation.

A whispering effect can be achieved by replacing the voiced component by random noise. Since the power spectrum of the voiced signal is a combination of the vocal tract and the glottal pulse, we would need to remove the spectral roll-off of the glottal pulse. This means that the power spectrum of the noise has to be high-pass in nature. Using white noise results in unnatural speech.

Voice conversion can be accomplished by altering the power spectrum [6, 28]. A warping transformation of the frequency scale can be achieved by shifting the LSF or the LPC roots, if using an LPC approach, or a warping curve if using an FFT representation.

Adding a controlled number of delayed and attenuated echoes can enhance an otherwise *dry* signal. If the delay is longer, it can simulate the room acoustics of a large hall.

16.6. Evaluation of TTS Systems

How do we determine whether one TTS system is better than another? Being able to evaluate TTS systems allows a customer to select the best system for his or her application. TTS evaluation is also important for developers of such systems to set some numerical goals in their design. As in any evaluation, we need to define a metric, which generally is dependent on the particular application for which the customer wants the TTS system. Such a metric consists of one or several variables of a system that are measured. Gibbon et al. [19] present a good summary of techniques used in evaluation of TTS systems.

Amazon/VB Assets
Exhibit 1012
Page 860

Here we present a taxonomy of a TTS evaluation:

- *Glass-box vs. black-box evaluation.* There are two types of evaluation of TTS systems according to whether we evaluate the whole system or just one of its components: black-box and glass-box. A black-box evaluation treats the TTS system as a black box and evaluates the system in the context of a real-world application. Thus, a system may do very well on a flight reservation application but poorly on an e-mail reading application. In a glass-box evaluation, we attempt to obtain diagnostics by evaluating the different components that make up a TTS system.
- *Laboratory vs. field.* We can also conduct the study in a laboratory or in the field. While the former is generally easier to do, the latter is generally more accurate.
- *Symbolic vs. acoustic level.* In general, TTS evaluation is normally done by analyzing the output waveform, the so-called acoustic level. Glass-box evaluation at the symbolic level is useful for the text analysis and phonetic module, for example.
- *Human vs. automated.* There are two fundamentally distinct ways of evaluating speech synthesizers, according to how a given attribute of the system is estimated. One is to use human subjects; the other to automate the evaluation process. Both types have some issues in common and a number of dimensions of systematic variation. But the fundamental distinction is one of cost. In system development, and particularly in research on high-quality systems, it can be prohibitively expensive to run continuously a collection of human assessments of every algorithmic change or idea. Though human-subject checkpoints are needed throughout the development process, human testing is of greatest importance for the integrated, functionally complete system in the target field setting. At all earlier stages of development, automated testing should be substituted for human-subject testing wherever possible. The hope is that someday TTS research can be conducted as ASR research is today: algorithms are checked for accuracy or performance improvements automatically in the lab, while human subjects are mainly used when the final integrated system is deployed for field testing. This allows for rapid progress in the basic algorithms contributing to accuracy on any given dimension.
- *Judgment vs. functional testing.* Judgment tests are those that measure the TTS system in the context of the application where it is used, such as what percentage of the time users hang up an IVR system. System *A* may be more appropriate than system *B* for a banking application where most of the speech consists of numerical values, and system *B* may be better than system *A* for reading e-mail over the phone. Nonetheless, it is useful to use functional tests that measure task-independent variables of a TTS system, since such tests allow an easier comparison among different systems, albeit a nonoptimal one.

Amazon/VB Assets
Exhibit 1012
Page 861

Since a human listener is the consumer of a TTS system, tests have been designed to determine the following characteristics of synthesized speech: intelligibility, overall quality, naturalness, suitability for a given task, and pleasantness. In addition, testing has been used for ranking and comparing a number of competing speech synthesizers, and for comparing synthetic with natural speech.

- *Global vs. analytic assessment.* The tests can measure such global aspects as overall quality, naturalness, and acceptability. Analytic tests can measure the rate, the clarity of vowels and consonants, the appropriateness of stresses and accents, pleasantness of voice quality, and tempo. Functional tests have been designed to test the intelligibility of individual sounds (phoneme monitoring), of combinations of sounds (syllable monitoring), and of whole words (word monitoring) in isolation as well as in various types of context.

It should be noted that all the above tests focus on segments, words, and sentences. This is a historical artifact, and as the field evolves, we should see an emphasis on testing of higher-order units. The diagnostic categories mentioned above can be used as a basis for developing tests of systems that take other structure into account. Such systems might include document-to-speech, concept-to-speech, and simulated conversation or dialog. A good system will reflect document and paragraph structure in the pausing and rhythm. Concept-to-speech systems claim to bring fuller knowledge of the intended use of information to bear in message generation and synthesis. Simulated dialog systems, or human-computer dialog systems, have to mimic a more spontaneous style, which is a subtle quality to evaluate. The tricky issue with higher-order units is the difficulty of simple choice or transcription-oriented measures. To develop tests of higher-order synthesizers, the word and sentence metrics can be applied to components and the overall system until reasonable intelligibility can be verified. Then tests of the special issues raised by higher-order systems can be conducted. Appropriate measures might be MOS overall ratings, preference between systems, summarization/gist transcription with subjective scoring, and task-based measures such as following directions. With task-based testing of higher-order units, both the correctness of direction-following and the time to completion, an indirect measure of intelligibility, pleasantness, and fatigue, can be recorded.

Furthermore, speech perception is not simply auditory. As discussed in Chapter 2, the McGurk effect [36] shows that perception of a speech sound is heavily influenced by visual cues. Synthetic speech is thus perceived with higher quality when a talking head is added as a visual cue [9, 18].

Glass-box evaluation of the text analysis and phonetic analysis modules, requiring evaluation at the *symbolic* level, is done in Chapter 14. A glass-box evaluation of the prosody module is presented in Chapter 15. In this section we include glass-box evaluation of the synthesis module, as well as a black-box evaluation of the whole system.

Amazon/VB Assets
Exhibit 1012
Page 862

16.6.1. Intelligibility Tests

A critical measurement of a TTS system is whether or not human listeners can understand the text read by the system. Tests that measure this are called *intelligibility tests*. In this section we describe the Diagnostic Rhyme Test, the Modified Rhyme Test, the Phonetically Balanced word list test, the Haskins Syntactic Sentence Test, and the Semantically Unpredictable Sentence Test. The first three were described in a procedure approved by the American National Standards Institute [5].

Among the best known and most mature of these tests is the Diagnostic Rhyme Test (DRT) proposed by Voiers [54], which provides for diagnostic and comparative evaluation of the intelligibility of single initial consonants. The test runs twice through the list of 96 rhyming pairs shown in Table 16.13. The test consists of identification choice between two alternative English (or target-language) words, differing by a single phonetic feature in the initial consonant. For English the test includes contrasts among easily confusable paired consonant sounds such as *veal/feel, meat/beat, fence/pence, cheep/keep, weed/reed,* and *hit/fit.* In the test, both *veal* and *feel* are presented with the response alternatives *veal* and *feel.* Six contrasts are represented, namely voicing, nasality, sustention, sibilation, graveness, and compactness. Each contrast is included 32 times in the test, combined with 8 different vowels. The percentage of right answers is used as an indicator of speech synthesizer intelligibility. The tests use a minimum of five talkers and five listeners; larger subject groups reduce the margin of error. Even for high-quality speech coders, 100% *correct* responses are rarely achieved, so synthesizer results should be interpreted generously.

Table 16.13 The 192 stimulus words of the Diagnostic Rhyme Test (DRT).

Voicing		Nasality		Sustenation		Sibilation		Graveness		Compactness	
veal	feel	meat	beat	vee	bee	zee	thee	weed	reed	yield	wield
bean	peen	need	deed	sheet	cheat	cheep	keep	peak	teak	key	tea
gin	chin	mitt	bit	vill	bill	jilt	gilt	bid	did	hit	fit
dint	tint	nip	dip	thick	tick	sing	thing	fin	thin	gill	dill
zoo	sue	moot	boot	foo	pooh	juice	goose	moon	noon	coop	poop
dune	tune	news	dues	shoes	choose	chew	coo	pool	tool	you	rue
vole	foal	moan	bone	those	doze	joe	go	bowl	dole	ghost	boast
goat	coat	note	dote	though	dough	sole	thole	fore	thor	show	so
zed	said	mend	bend	then	den	jest	guest	met	net	keg	peg
dense	tense	neck	deck	fence	pence	chair	care	pent	tent	yen	wren
vast	fast	mad	bad	than	dan	jab	gab	bank	dank	gat	bat
gaff	calf	nab	dab	shad	chad	sank	thank	fad	thad	shag	sag
vault	fault	moss	boss	thong	tong	jaws	gauze	fought	thought	yawl	wall
daunt	taunt	gnaw	daw	shaw	chaw	saw	thaw	bong	dong	caught	thought
jock	chock	mom	bomb	von	bon	jot	got	wad	rod	hop	fop
bond	pond	knock	dock	vox	box	chop	cop	pot	tot	got	dot

Amazon/VB Assets
Exhibit 1012
Page 863

A variant of this is the Modified Rhyme Test (MRT) proposed by House [22], which also uses a 300-entry word list for subjective intelligibility testing. The modified Rhyme Test (shown in Table 16.14) uses 50 six-word lists of rhyming or similar-sounding monosyllabic English words, e.g., *went, sent, bent, dent, tent, rent.* Each word is basically Consonant-Vowel-Consonant (with a few consonant clusters), and the six words in each list differ only in the initial or final consonant sound(s). Listeners are asked to identify which of the words was spoken by the synthesizer (closed response), or in some cases to enter any word they thought they heard (open response). A carrier sentence, such as "Would you write <test word> now," is usually used for greater naturalness in stimulus presentation. Listener responses can be scored as the number of words heard correctly; or the frequency of confusions of particular consonant sounds. This can be viewed as *intelligibility* of the synthesizer.

Though this is a nice isolation of one property, and as such is particularly appropriate for diagnostic use, it is not intended to substitute for fuller evaluation under more realistic listening conditions involving whole sentences. Segmental intelligibility is somewhat overestimated in these tests, because all the alternatives are real words and the subjects can adjust their perception to match the closest word. A typical human voice gives an MRT score of about 99%, with that of TTS systems generally ranging from 70% to 95%.

Table 16.14 The 300 stimulus words of the Modified Rhyme Test (MRT).

went	sent	bent	dent	tent	rent	same	name	game	tame	came	fame
hold	cold	told	fold	sold	gold	peel	reel	feel	eel	keel	heel
pat	pad	pan	path	pack	pass	hark	dark	mark	bark	park	lark
lane	lay	late	lake	lace	lame	heave	hear	heat	heal	heap	heath
kit	bit	fit	hit	wit	sit	cup	cut	cud	cuff	cuss	cud
must	bust	gust	rust	dust	just	thaw	law	raw	paw	jaw	saw
teak	team	teal	teach	tear	tease	pen	hen	men	then	den	ten
din	dill	dim	dig	dip	did	puff	puck	pub	pus	pup	pun
bed	led	fed	red	wed	shed	bean	beach	beat	beak	bead	beam
pin	sin	tin	fin	din	win	heat	neat	feat	seat	meat	beat
dug	dung	duck	dud	dub	dun	dip	sip	hip	tip	lip	rip
sum	sun	sung	sup	sub	sud	kill	kin	kit	kick	king	kid
seep	seen	seethe	seek	seem	seed	hang	sang	bang	rang	fang	gang
not	tot	got	pot	hot	lot	took	cook	look	hook	shook	book
vest	test	rest	best	west	nest	mass	math	map	mat	man	mad
pig	pill	pin	pip	pit	pick	ray	raze	rate	rave	rake	race
back	bath	bad	bass	bat	ban	save	same	sale	sane	sake	safe
way	may	say	pay	day	gay	fill	kill	will	hill	till	bill
pig	big	dig	wig	rig	fig	sill	sick	sip	sing	sit	sin
pale	pace	page	pane	pay	pave	bale	gale	sale	tale	pale	male
cane	case	cape	cake	came	cave	wick	sick	kick	lick	pick	tick
shop	mop	cop	top	hop	pop	peace	peas	peak	peach	peat	peal
coil	oil	soil	toil	boil	foil	bun	bus	but	bug	buck	buff
tan	tang	tap	tack	tam	tab	sag	sat	sass	sack	sad	sap
fit	fib	fizz	fill	fig	fin	fun	sun	bun	gun	run	nun

Amazon/VB Assets
Exhibit 1012
Page 864

The set of twenty phonetically balanced (PB) word lists was developed during World War II and has been used very widely since then in subjective intelligibility testing. In Table 16.15 we include the first four PB word lists [20]. The words in each list are presented in a new, random order each time the list is used, each spoken in the same carrier sentence. The PB intelligibility test requires more training of listeners and talkers than other subjective tests and is particularly sensitive to SNR: a relatively small change causes a large change in the intelligibility score.

Tests using the *Haskins Syntactic Sentences* [40] go somewhat farther toward more realistic and holistic stimuli. This test set consists of 100 semantically unpredictable sentences of the form *The <Adjective> <Noun1> <Verb> the <Noun2>*, such as *"The old farm cost the blood,"* using high-frequency words. Compared with the rhyme tests, contextual predictability based on meaning is largely lacking, the longer speech streams are more realistic, and more coarticulation is present. Intelligibility is indicated by percentage of words correct.

Another test minimizing predictability is *Semantically Unpredictable Sentences* [23], with test sets for Dutch, English, French, German, Italian, and Swedish. A short template of syntactic categories provides a frame, into which words are randomly slotted from the lexicon. For example, the template *<Subject> <Verb> <Adverbial>* might appear as *"The chair ate through the green honesty."* Fifty sentences (10 per syntactic template) are considered adequate to test a synthesizer. Open transcription is requested, and *sentences correct* is used to score a synthesizer's intelligibility. Other such tests exist, and some include systematic variation of prosody on particular words or phrases as well.

The Harvard Psychoacoustic Sentences [16] is a set of 100 meaningful, syntactically varied, phonetically balanced sentences, such as *"Add salt before you fry the egg,"* requiring an *open response identification,* instead of a multiple-choice test.

Table 16.15 Phonetically balanced word lists.

List 1	are, bad, bar, bask, box, cane, cleanse, clove, crash, creed, death, deed, dike, dish, end, feast, fern, folk, ford, fraud, fuss, grove, heap, hid, hive, hunt, is, mange, no, nook, not, pan, pants, pest, pile, plush, rag, rat, ride, rise, rub, slip, smile, strife, such, then, there, toe, use, wheat
List 2	awe, bait, bean, blush, bought, bounce, bud, charge, cloud, corpse, dab, earl, else, fate, five, frog, gill, gloss, hire, hit, hock, job, log, moose, mute, nab, need, niece, nut, our, perk, pick, pit, quart, rap, rib, scythe, shoe, sludge, snuff, start, suck, tan, tang, them, trash, vamp, vast, ways, wish
List 3	ache, air, bald, barb, bead, cape, cast, check, class, crave, crime, deck, dig, dill, drop, fame, far, fig, flush, gnaw, hurl, jam, law, leave, lush, muck, neck, nest, oak, path, please, pulse, rate, rouse, shout, sit, size, sob, sped, stag, take, thrash, toil, trip, turf, vow, wedge, wharf, who, why
List 4	bath, beast, bee, blonde, budge, bus, bush, cloak, course, court, dodge, dupe, earn, eel, fin, float, frown, hatch, heed, hiss, hot, how, kite, merge, lush, neat, new, oils, or, peck, pert, pinch, pod, race, rack, rave, raw, rut, sage, scab, shed, shin, sketch, slap, sour, starve, strap, test, tick, touch

Amazon/VB Assets
Exhibit 1012
Page 865

16.6.2. Overall Quality Tests

While a TTS system has to be intelligible, this does not guarantee user acceptance, because its quality may be far from that of a human speaker. In this section we describe the Mean Opinion Score and the Absolute Category Ratings.

Human-subject judgment testing for TTS can adapt methods from speech-coding evaluation (see Chapter 7). With speech coders, *Mean Opinion Score* (MOS) is administered by asking 10 to 30 listeners to rate several sentences of coded speech on a scale of 1 to 5 (1 = Bad, 2 = Poor, 3 = Fair, 4 = Good, 5 = Excellent). The scores are averaged, resulting in an overall MOS rating for the coder. This kind of methodology can be applied to speech synthesizers as well. Of course, as with any human subject test, it is essential to carefully design the listening situation and carefully select the subject population, controlling for education, experience, physiological disorders, dialect, etc. As with any statistically interpreted test, the standard analyses of score distributions, standard deviation, and confidence intervals must be performed. The range of quality in coder evaluations by MOS are shown in Table 16.16.

Since we are making the analogy to coders, certain ironies can be noted. Note the lowest-range descriptor for coder evaluation: *synthetic*. In using MOS for synthesis testing, output is being evaluated by implicit reference to real human speech, and the upper range in the coder MOS interpretations above (3.5–4.5) is probably not applicable to the output of most TTS systems. Even a *good* TTS system might fare poorly on such a coder MOS evaluation. Therefore, the MOS interpretive scale, when applied to synthesis, cannot be absolute as the above coding-based interpretive table would imply. Furthermore, subjects participating in MOS-like tests of synthesizers should be made aware of the special nature of the speech (synthetic) and adjust their expectations accordingly. Finally, no matter how carefully the test is designed and administered, it is difficult to correlate, compare, and scale such measures. Nevertheless, MOS tests are perhaps suited to relative ranking of various synthesizers. The 1-to-5 scale is categorical, but similar judgment tests can be run in *magnitude* mode, with the strength of the quality judgment being indicated along a continuous scale, such as a moving slider bar.

Table 16.16 Mean opinion score (MOS) ratings and typical interpretations.

MOS Scores	Quality	Comments
4.0–4.5	Toll/Network	Near-transparent, "in-person" quality
3.5–4.0	Communications	Natural, highly intelligible, adequate for telecommunications, changes and degradation of quality very noticeable
2.5–3.5	Synthetic	Usually intelligible, can be unnatural, loss of speaker recognizability, inadequate levels of naturalness

Amazon/VB Assets
Exhibit 1012
Page 866

Table 16.17 Listening Quality Scale.

Quality of the Speech	Score
Excellent	5
Good	4
Fair	3
Poor	2
Bad	1

The International Telecommunication Union (ITU) has attempted to specify some standards for assessing synthetic speech, including spliced digitized words and phrases, typically with the expectation of delivery over the phone. The *Absolute Category Rating* (ACR) system recommended by ITU P.800 offers instructions to be given to subjects for making category judgments in MOS-style tests of the type discussed here. The first is the *Listening Quality Scale*, shown in Table 16.17, and the second the *Listening Effort Scale* shown in Table 16.18.

It is sometimes possible to get subjects to pay particular attention to various particular features of the utterance, which may be called *analytic* as opposed to *global* listening. The desired features generally have to be described somehow, and these descriptions can be a bit vague. Thus, standard measures of reliability and validity, as well as result normalization, must be applied. Typical descriptors for important factors in analytic listening might be: *smoothness*, *naturalness*, *pleasantness*, *clarity*, *appropriateness*, etc., each tied to a particular underlying target quality identified by the system designers. For example, smoothness might be a descriptor used when new algorithms for segment concatenation and blending are being evaluated in a concatenative system. Naturalness might be the quality descriptor when a formant-based system has been made more natural by incorporation of a richer glottal source function. Some elements of the speech can be more directly identified to the subject in familiar terms. For example, pleasantness might be a way of targeting the pitch contours for attention, or the subject could be specifically asked to rank the pitch contours per se, in terms of naturalness, pleasantness, etc. Appropriateness might be a way of getting at judgments of accentuation: e.g., a stimulus that was accented as "... *birthday PARTY*" might be judged less appropriate, in a neutral semantic context, than one that was perceived as "... *BIRTHDAY party.*" But no matter how the attributes are described, in human-subject MOS-style testing there cannot be a clear and consistent separation of effects.

Table 16.18 Listening Effort Scale.

Effort Required to Understand the Meanings of Sentences	Score
Complete relaxation possible; no effort required	5
Attention necessary; no appreciable effort required	4
Moderate effort required	3
Considerable effort required	2
No meaning understood with any feasible effort	1

Amazon/VB Assets
Exhibit 1012
Page 867

16.6.3. Preference Tests

Normalized MOS scores for different TTS systems can be obtained without any direct preference judgments. If direct comparisons are desired, especially for systems that are informally judged to be fairly close in quality, another ITU recommendation, the *Comparison Category Rating* (CCR) method, may be used. In this method, listeners are presented with a pair of speech samples on each trial. The order of the *system A system B* samples is chosen at random for each trial. On half of the trials, the *system A* sample is followed by the *system B* sample. On the remaining trials, the order is reversed. Listeners use the instructions in Table 16.19 to judge the quality of the second sample relative to that of the first. Sometimes the granularity can be reduced as much as simply "*prefer A/prefer B.*"

Assuming (A,B) is the reference presentation order, scores for the (B,A) presentations may be normalized by reversing their signs (e.g., –1 in B,A order becomes 1, etc.). Subsequently, standard statistical summarizations may be performed, like the one described in Chapter 3.

Table 16.19 Preference ratings between two systems. The quality of the second utterance is compared to the quality of the first by means of 7 categories. Sometimes only better, same, or worse are used.

3	Much Better
2	Better
1	Slightly Better
0	About the Same
-1	Slightly Worse
-2	Worse
-3	Much Worse

16.6.4. Functional Tests

Functional testing places the human subject in the position of carrying out some task related to, or triggered by, the speech. This can simulate a full field deployment, with a usercentric task, or can be more of a laboratory situation, with a testcentric task. In the laboratory situation, various kinds of tasks have been proposed. In *analytic* mode, functional testing can enforce isolation of the features to be attended to in the structure of the test stimuli themselves. This can lead to a more precise form of result than the MOS judgment approach. There have been a wide variety of proposals and experiments of this type.

One of the well-known facts in TTS evaluation is that the quality of a system is dominated by the quality of its worst component. While it may be argued that it is impossible to separate the effects of the front-end analysis and back-end synthesis, it is convenient to do so to gain a better understanding of each component. An attempt to study the quality of the speech synthesis module has been done via the use of natural instead of synthetic prosody. This way, it is presumed that the prosody module is doing the best possible job, and that any

Amazon/VB Assets
Exhibit 1012
Page 868

problem is then due to a deficient speech synthesis. The natural pitch contour can be obtained with a pitch tracker (or using a laryngograph signal), and the natural durations can be obtained either through manual segmentation or through the use of a speech recognition system used in forced-alignment mode. Plumpe and Meredith [44] conducted a preference test between original recordings and waveforms created when one of the modules of a concatenative TTS system used synthetically generated values instead of the natural values. The results indicated that using synthetic pitch instead of natural pitch was the cause of largest degradation according to listeners, and, thus, that pitch generation was the largest bottleneck in the system. The pitch-generation module was followed by the spectral discontinuities at the concatenation points, with duration being the least damaging.

Some functional tests are much more creative than simple transcription, however. They could, in theory, border on related areas, such as memory testing, involving summarizing passages, or following synthesized directions, such as a route on a map. The ultimate test of synthesis, in conjunction with all other language interface components, is said to be the Turing test [53]. In this amusing scenario, a human being is placed into conversation with a computational agent, represented vocally for our purposes, perhaps over the telephone. As Turing put it: "It is proposed that a machine may be deemed intelligent, if it can act in such a manner that a human cannot distinguish the machine from another human merely by asking questions via a mechanical link." Turing predicted that in the future "an average interrogator will not have more than a 70 percent chance of making the right identification, human or computer on the other end, after five minutes of questioning" in this game. A little reflection might raise objections to this procedure as a check on speech output quality per se, since some highly intelligent people have speech disabilities, but the basic idea should be clear, and it remains a Holy Grail for the artificial intelligence field generally. Of course, no automated or laboratory test can substitute for a real-world trial with paying customers.

16.6.5. Automated Tests

The tests described above always involved the use of human subjects and are the best tests that can be used to evaluate a TTS system. Unfortunately, they are time consuming and expensive to conduct. This limits their application to an infrequent use, which can hardly have any diagnostic value. Automated *objective* tests usually involve establishing a test corpus of correctly tagged examples of the tested phenomena, which can be automatically checked. This style of testing is particularly appropriate when working with isolated components of the TTS system, for diagnosis or regression testing (glass-box testing). It is not particularly productive to discuss such testing in the abstract, as the test features must closely track each system's design and implementation. Nevertheless, a few typical areas for testing can be noted. In general, tests are simultaneously testing the linguistic model and content as well as the software implementation of a system, so whenever a discrepancy arises, both possible sources of error must be considered.

Several automated tests for text analysis and letter-to-sound conversion are presented in Chapter 14. A number of automated tests for prosody are discussed in Chapter 15. Here we touch on automated tests for the synthesis module.

Amazon/VB Assets
Exhibit 1012
Page 869

The ITU has created the P.861 proposal for estimating perceptual scores using automated signal-based measurements. The P.861 specifies a particular technique known as *Perceptual Speech Quality Measurement* (PSQM). In this method, for each analysis frame, various quantified measures based on the time signal, the power spectrum, the Bark power spectrum, the excitation spectrum, the compressed loudness spectrum, etc. of both the reference and the test signal can be computed. In some cases the PSQM score can be converted to an estimated MOS score, with interpretations similar to those of Table 16.16. At present such methods are limited primarily to analysis of telephone-quality speech (300–3400 Hz bandwidth), to be compared with closely related reference utterances. This method could perhaps be adapted to stand in for human judgments during system development of new versions of modules, say glottal source functions in a formant synthesizer, comparing the resulting synthetic speech to a standard reference system's output on a given test sample.

16.7. HISTORICAL PERSPECTIVE AND FURTHER READING[3]

In 1779 in St. Petersburg, Russian Professor Christian Kratzenstein explained physiological differences between five long vowels (*/a/*, */e/*, */i/*, */o/*, and */u/*) and made apparatus to produce them artificially. He constructed acoustic resonators similar to the human vocal tract and activated the resonators with vibrating reeds as in music instruments. Von Kempelen (1734–1804) proposed in 1791 in Vienna a mechanical speaking machine that could produce not just vowels but whole words and sentences (see Figure 16.19). While working with his speaking machine, he demonstrated a speaking chess-playing machine. Unfortunately, the main mechanism of the machine was a concealed, legless chess-player expert. Therefore, his real speaking machine was not taken as seriously as it should have been. In 1846, Joseph Faber developed a synthesizer, called speech organ, that had more control of pitch to the extent it could sing *God Save the Queen* in a performance in London.

The first electrical synthesis device was introduced by Stewart in 1922 [4]. The device had a buzzer as excitation and two resonant circuits to model the acoustic resonances of the vocal tract and was able to generate single static vowel sounds with the first two formants. In 1932 Japanese researchers Obata and Teshima added a third formant for more intelligible vowels.

Homer Dudley of Bell Laboratories demonstrated at the 1939 New York World's Fair the *Voder*, the first electrical speech synthesizer, which was human-controlled. The operator worked at a keyboard, with a wrist bar to control the voicing parameter and a pedal for pitch control (see Figure 16.20 and Figure 16.21), and it was able to synthesize continuous speech. The *Pattern Playback* is an early talking machine that was built by Franklin S. Cooper and his colleagues at Haskins Laboratories in the late 1940s. This device synthesized sound by passing light through spectrograms that in turn modulated an oscillator with a fixed F0 of 120 Hz and 50 harmonics.

[3] Chapter 6 includes a historical perspective on representation of speech signals that is intimately tied to speech synthesis.

Amazon/VB Assets
Exhibit 1012
Page 870

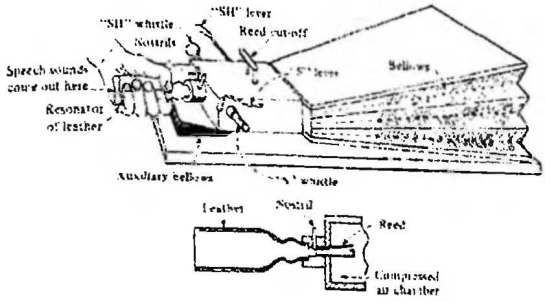


Figure 16.19 Wheatstone's reconstruction of von Kempelen's speaking machine [14] (after Flanagan [17]).

The first analog parallel formant synthesizer, the *Parametric Artificial Talker* (PAT), was developed in 1953 by Walter Lawrence of the Signals Research and Development Establishment of the British Government. Gunnar Fant of the KTH in Sweden developed an analog cascade formant synthesizer, the OVE II. Both Lawrence and Fant showed in 1962 that by manually tuning the parameters, a natural sentence could be reproduced reasonably faithfully. Acoustic analog synthesizers were also known as terminal analogs, resonance-synthesizers. John Holmes tuned by hand the parameters of his formant synthesizer so well that the average listener could not tell the difference between the synthesized sentence "*I enjoy the simple life*" and the natural one [31].

Figure 16.20 The Voder developed by Homer Dudley of Bell Labs at the 1939 World's Fair in New York. The operator worked at a keyboard, with a wrist bar to control the voicing parameter and a pedal for pitch control.

Amazon/VB Assets
Exhibit 1012
Page 871

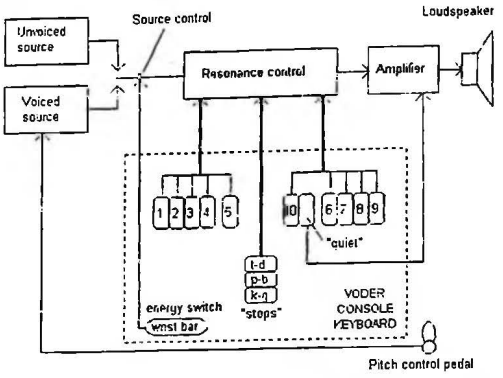


Figure 16.21 Block diagram of the Voder by Homer Dudley, 1939 (after Flanagan [17]).

The first articulatory synthesizer, the DAVO, was developed in 1958 by George Rosen at M.I.T. Cecil Coker designed rules to control a low-dimensionality articulatory model in 1968. Paul Mermelstein and James Flanagan from Bell Labs also used articulatory synthesis in 1976. Articulatory synthesis, however, never took off, because formant synthesis was better understood at the time.

The advent of the digital computer prompted John Kelly and Louis Gerstman to create in 1961 the first phonemic-synthesis-by-rule program. John Holmes and his colleagues Ignatius Mattingly and John Shearme developed a rule program for a formant synthesizer at JSRU in England. The first full text-to-speech system was developed by Noriko Umeda in 1968 at the Electrotechnical Laboratory of Japan. It was based on an articulatory model and included a syntactic analysis module with sophisticated heuristics. The speech was quite intelligible, but monotonous and far from the quality of present systems.

In 1976, Raymond Kurzweil developed a unique reading machine for the blind, a computer-based device that read printed pages aloud. It was an 80-pound device that shot a beam of light across each printed page, converting the reflected light into digital data that was transformed by a computer into synthetic speech. It made reading of all printed material possible for blind people, whose reading has previously been limited to material translated into Braille. The work of Dennis Klatt of MIT had a large influence in the field. In 1979 together with Jonathan Allen and Sheri Hunnicut he developed the MITalk system. Two years later Klatt introduced his famous Klattalk system, which used a new sophisticated voicing source.

The early 1980s marked the beginning of commercial TTS systems. The Klattalk system was the basis of Telesensory Systems' Prose-2000 commercial TTS system in 1982. It also formed the basis for Digital Equipment Corporation's DECtalk commercial system in 1983, probably the most widely used TTS system of the twentieth century. The Infovox TTS system, the first multilanguage formant synthesizer, was developed in Sweden by Rolf Carlson, Bjorn Granstrom, and Sheri Hunnicutt in 1982, and it was a descendant of Gunnar Fant's OVE system. The first integrated circuit for speech synthesis was probably the Votrax chip, which consisted of cascade formant synthesizer and simple low-pass smoothing

Amazon/VB Assets
Exhibit 1012
Page 872

circuits. In 1978 Richard Gagnon introduced an inexpensive Votrax-based Type-n-Talk system. The first work in concatenative speech synthesis was done in 1968 by Rex Dixon and David Maxey, where diphones were parametrized with formant frequencies and then concatenated. In 1977, Joe Olive and his colleagues at Bell Labs [41] concatenated linear-prediction diphones. In 1982 Street Electronics introduced the Echo system, a diphone concatenation synthesizer which was based on a newer version of the same chip as in the Speak-n-Spell toy introduced by Texas Instruments in 1980.

Concatenative systems started to gain momentum in 1985 with the development of the PSOLA prosody modification technique by France Telecom's Charpentier and Moulines. PSOLA increased the text coverage of concatenative systems by allowing diphones to have their prosody modified. The hybrid Harmonic/Stochastic (H/S) model of Abrantes [1] has also been successfully used for prosody modification. The foundation of corpus-based concatenative systems was developed by a team of researchers at ATR in Japan in the early 1990s [10, 27]. The use of a large database of long units was also pioneered by researchers at AcuVoice Inc. Other corpus-based systems have made use of HMMs to automatically segment speech databases, as well as to serve as units in concatenative synthesis [13, 24]. Microsoft integrated a concatenative TTS [24] in Windows 2000.

For more detailed description of speech synthesis development and history see, for example, [31] and [17] and references in these. A number of audio clips are available in Klatt [31] showing the progress through the early years. You can hear samples at the Smithsonian's Speech Synthesis History Project [35]. A Web site with comparison of recent TTS systems can be found at [33].

REFERENCES

[1] Abrantes, A.J., J.S. Marques, and I.M. Trancoso. "Hybrid Sinusoidal Modeling of Speech without Voicing Decision," *Proc. Eurospeech*, 1991, Genoa, Italy, pp. 231-234.

[2] Acero, A., "Source-Filter Models for Time-Scale Pitch-Scale Modification of Speech," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 881-884.

[3] Acero, A., "Formant Analysis and Synthesis Using Hidden Markov Models," *Eurospeech*, 1999, Budapest, pp. 1047-1050.

[4] Allen, J., M.S. Hunnicutt, and D.H. Klatt. *From Text to Speech: The MITalk System*, 1987, Cambridge, UK, University Press.

[5] ANSI, *Method for Measuring the Intelligibility of Speech Over Communication Systems*, 1989, American National Standards Institute.

[6] Arslan, L.M. and D. Talkin, "Speaker Transformation Using Sentence HMM Based Alignments and Detailed Prosody Modification," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, pp. 289-292.

[7] Beutnagel, M., *et al.*, "The AT&T Next-Gen TTS System," *Joint Meeting of ASA*, 1999, Berlin, pp. 15-19.

Amazon/VB Assets
Exhibit 1012
Page 873

[8] Bickley, C.A., K.N. Stevens, and D.R. Williams, "A Framework for Synthesis of Segments Based on Pseudoarticulatory Parameters" in *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag, pp. 211-220.

[9] Bregler, C., M. Covell, and M. Slaney, "Video Rewrite: Driving Visual Speech with Audio," *ACM Siggraph*, 1997, Los Angeles, pp. 353-360.

[10] Campbell, W.N. and A.W. Black, "Prosody and the Selection of Source Units for Concatenative Synthesis" in *Progress in Speech Synthesis*, J.V. Santen, *et al.*, eds., 1996, pp. 279-292, Springer Verlag.

[11] Covell, M., M. Withgott, and M. Slaney, "Mach1: Nouniform Time-Scale Modification of Speech," *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 349-352.

[12] Crochiere, R., "A Weighted Overlap-Add Method of Short Time Fourier Analysis/Synthesis," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1980, **28**(2), pp. 99-102.

[13] Donovan, R. and P. Woodland, "Improvements in an HMM-based Speech Synthesizer," *Proc. of the EuroSpeech Conf.*, 1995, Madrid, pp. 573-576.

[14] Dudley, H. and T.H. Tarnoczy, "The Speaking Machine of Wolfgang von Kempelen," *Journal of the Acoustical Society of America*, 1950, **22**, pp. 151-166.

[15] Dutoit, T., *An Introduction to Text-to-Speech Synthesis*, 1997, Kluwer Academic Publishers.

[16] Egan, J., "Articulation Testing Methods," *Laryngoscope*, 1948, **58**, pp. 955-991.

[17] Flanagan, J., *Speech Analysis, Synthesis and Perception*, 1972, New York, Springer-Verlag.

[18] Galanes, F.G., *et al.*, "Generation of Lip-Synched Synthetic Faces from Phonetically Clustered Face Movement Data," *AVSP*, 1998, Terrigal, Australia.

[19] Gibbon, D., R. Moore, and R. Winski, *Handbook of Standards and Resources for Spoken Language Systems*, 1997, Berlin & New York, Walter de Gruyter Publishers.

[20] Goldstein, M., "Classification of Methods Used for Assesment of Text-to-Speech Systems According to the Demands Placed on the Listener," *Speech Communication*, 1995, **16**, pp. 225-244.

[21] Hon, H., *et al.*, "Automatic Generation of Synthesis Units for Trainable Text-to-Speech Systems," *Int. Conf. on Acoustics, Signal and Speech Processing*, 1998, Seattle, WA, pp. 293-296.

[22] House, A., *et al.*, "Articulation Testing Methods: Consonantal Differentiation with a Closed Response Set," *Journal of the Acoustical Society of America*, 1965, **37**, pp. 158-166.

[23] Howard-Jones, P., *SOAP, Speech Output Assessment Package*, 1992, ESPRIT SAM-UCL-042.

[24] Huang, X., *et al.*, "Whistler: A Trainable Text-to-Speech System," *Int. Conf. on Spoken Language Processing*, 1996, Philadephia, pp. 2387-2390.

Amazon/VB Assets
Exhibit 1012
Page 874

[25] Huang, X., *et al.*, "Recent Improvements on Microsoft's Trainable Text-To-Speech System - Whistler," *Int. Conf. on Acoustics, Signal and Speech Processing*, 1997, Munich, Germany, pp. 959-962.

[26] Hunt, A.J. and A.W. Black, "Unit Selection in a Concatenative Speech Synthesis System Using a Large Speech Database," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, pp. 373-376.

[27] Iwahashi, N., N. Kaiki, and Y. Sagisaka, "Concatenation Speech Synthesis by Minimum Distortion Criteria," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, pp. 65-68.

[28] Kain, A. and M. Macon, "Text-to-Speech Voice Adaptation from Sparse Training Data," *Int. Conf. on Spoken Language Systems*, 1998, Sydney, Australia, pp. 2847-2850.

[29] Klatt, D. and L. Klatt, "Analysis, Synthesis and Perception of Voice Quality Variations among Female and Male Talkers," *Journal of the Acoustical Society of America*, 1990, **87**, pp. 737-793.

[30] Klatt, D.H., "Software for a Cascade/Parallel Formant Synthesizer," *Journal of Acoustical Society of America*, 1980, **67**, pp. 971-995.

[31] Klatt, D.H., "Review of Text to Speech Conversion for English," *Journal of the Acoustical Society of America*, 1987, **82**, pp. 737-793.

[32] Krishnamurthy, A.K. and D.G. Childers, "Two Channel Speech Analysis," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1986, **34**, pp. 730-743.

[33] LDC, *Interactive Speech Synthesizer Comparison Site*, 2000, http://morph.ldc.upenn.edu.

[34] Maachi, M., "Coverage of Names," *Journal of the Acoustical Society of America*, 1993, **94**(3), pp. 1842.

[35] Maxey, H., *Smithsonian Speech Synthesis History Project*, 2000, http://www.mindspring.com/~dmaxey/ssshp/.

[36] McGurk, H. and J. MacDonald, "Hearing Lips and Seeing Voices," *Nature*, 1976, **264**, pp. 746-748.

[37] Mermelstein, P. and M.R. Schroeder, "Determination of Smoothed Cross-Sectional Area Functions of the Vocal Tract from Formant Frequencies," *Fifth Int. Congress on Acoustics*, 1965.

[38] Moulines, E. and F. Charpentier, "Pitch-Synchronous Waveform Processing Techniques for Text-to-Speech Synthesis Using Diphones," *Speech Communication*, 1990, **9**(5), pp. 453-467.

[39] Moulines, E. and W. Verhelst, "Prosodic Modifications of Speech" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Elsevier, pp. 519-555.

[40] Nye, P. and J. Gaitenby, *The Intelligibility of Synthetic Monosyllabic Words in Short, Syntactically Normal Sentences*, 1974, Haskins Laboratories.

[41] Olive, J., "Rule Synthesis of Speech from Dyadic Units," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1977, Hartford, CT, pp. 568-570.

[42] Olive, J.P., A. Greenwood, and J.S. Coleman, *Acoustics of American English Speech: a Dynamic Approach*, 1993, New York, Springer-Verlag.

Amazon/VB Assets
Exhibit 1012
Page 875

[43] Oliveira, L., "Estimation of Source Parameters by Frequency Analysis," *Proc. of the Eurospeech Conf.*, 1993, Berlin, pp. 99-102.

[44] Plumpe, M. and S. Meredith, "Which Is More Important in a Concatenative Text-to-Speech System: Pitch, Duration, or Spectral Discontinuity," *Third ESCA/COCOSDA Int. Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, pp. 231-235.

[45] Roucos, S. and A. Wilgus, "High Quality Time-Scale Modification of Speech," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1985, pp. 493-496.

[46] Sagisaka, Y. and N. Iwahashi, "Objective Optimization in Algorithms for Text-to-Speech Synthesis" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds., 1995, pp. 685-706, Elsevier.

[47] Santen, J.V., "Combinatorial Issue in Text-to-Speech Synthesis," *Proc. of the Eurospeech Conf.*, 1997, Rhodes, Greece, pp. 2511-2514.

[48] Sproat, R., *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, 1998, Dordrecht, Kluwer Academic Publishers.

[49] Stylianou, Y., "Assessment and Correction of Voice Quality Variabilities in Large Speech Databases for Concatenative Speech Synthesis," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1999, Phoenix, AZ, pp. 377-380.

[50] Stylianou, Y., J. Laroche, and E. Moulines, "High Quality Speech Modification Based on a Harmonic + Noise Model," *Proc. Eurospeech*, 1995, Madrid.

[51] Syrdal, A., A. Conkie, and Y. Stylianou, "Exploration of Acoustic Correlates in Speaker Selection for Concatenative Synthesis," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2743-2746.

[52] Syrdal, A., *et al.*, "Voice Selection for Speech Synthesis," *Journal of the Acoustical Society of America*, 1997, **102**, pp. 3191.

[53] Turing, A.M., "Computing Machinery and Intelligence," *Mind*, 1950, **LIX**(236), pp. 433-460.

[54] Voiers, W., A. Sharpley, and C. Hehmsoth, *Research on Diagnostic Evaluation of Speech Intelligibility*, 1975, Air Force Cambridge Research Laboratories, Bedford, MA.

[55] Yi, J., *Natural Sounding Speech Synthesis Using Variable-Length Units*, Masters' Thesis in EECS Dept. 1998, MIT, Cambridge, MA.

Amazon/VB Assets
Exhibit 1012
Page 876

PART V

SPOKEN LANGUAGE SYSTEMS

CHAPTER 17

Spoken Language Understanding

Formal methods for describing sentences are discussed in Chapter 11. While the context-free grammars and *n*-gram models have mathematically well-understood formulations and bounded processing complexity, they are only partial aids in interpreting semantic meaning of the sentences. Suppose a recognizer correctly transcribes a series of spoken words into the written form—the system still has no idea what to do, because there is often no direct mapping between a sequence of words (or the syntactic structure of the sentence) and the functions that the system provides. The problem can also be approached from the opposite direction, i.e., solving the recognition problem itself may require semantic analysis, or domain and language knowledge for perplexity reduction.

What is meant by *meaning* or *understanding*? We could define it operationally: understanding is when a computer we interact with understands our desires and delivers the goods. Or we could define it propositionally: the computer has an accurate and unambiguous representation of *who did what to whom* corresponding to a real-world situation. In practice, the concept of understanding is situation dependent, and both conceptions above have their places. Meaning is often a constellation that emerges from a conversational environment.

Amazon/VB Assets
Exhibit 1012
Page 879

There are four main interacting areas in *spoken language understanding* (SLU) systems from which meaning arises:

- **Intent**: goals of listener and speaker in the interaction
- **Context**: the pressures, opportunities, interruptions, etc. of the interaction scene and communication media
- **Content**: the propositional or literal content of each utterance and the discourse as a whole
- **Assumptions**: what each participant can assume about other participants' mental state, abilities, limitations, etc.

In this chapter we take a functional view of SLU systems, where the basic principle is to link linguistic expressions to concrete real-world entities. Currently, only with systems that are restricted to limited domains can understanding be attempted in practice. The domain restrictions allow the creation of specific, highly restricted language models and fully interpretable semantic descriptions that enable high accuracy and usability. Such systems are in contrast to speech recognition approaches that use large dictionaries, but make relatively *loose* or probabilistic predictions of word sequences for general dictation/transcription.

The need for spoken language understanding is double-edged. We generally want more than a string of word choices as a system's output. Instead, we want some interpretation of the word string that helps in accomplishing complex tasks. At the same time, being able to determine *what makes sense in context*, what is more or less likely as a speaker's input, could make a major contribution toward improving speech recognition word accuracy and search efficiency. SLU systems that combine the semantic precision of grammars with the probabilistic coverage of statistical language models can guide recognition and simultaneously control interpretation.

Figure 1.4 in Chapter 1 illustrates a basic SLU system architecture. The SLU problem can be broadly viewed as yet another pattern recognition problem. Namely, given a speech input $\mathbf{X}$, the objective of the system is to arrive at actions $\mathbf{A}$ (including dialog messages and necessary operations) so that the cost of choosing $\mathbf{A}$ is minimized. Assuming uniform cost, the optimal solution, known as the maximum a posteriori (MAP) decision, can be expressed as

$$\begin{aligned} \mathbf{A}^* &= \arg\max_{\mathbf{A}} P(\mathbf{A} \mid \mathbf{X}, S_{n-1}) \\ &\approx \arg\max_{\mathbf{A}, S_n} P(\mathbf{A} \mid S_n) \sum_{\mathbf{F}} P(S_n \mid \mathbf{F}, S_{n-1}) P(\mathbf{F} \mid \mathbf{X}, S_{n-1}) \end{aligned} \tag{17.1}$$

where $\mathbf{F}$ denotes semantic interpretation of $\mathbf{X}$ and S_n, the discourse semantics for the n^{th} dialog turn.

Based on the formulation in Eq. (17.1), a dialog system consists of three pattern recognition components:

- **Semantic parser**—use semantic model $P(\mathbf{F} \mid \mathbf{X}, S_{n-1})$ to convert $\mathbf{X}$ into a collection of semantic objects $\mathbf{F}$. This component is often further decomposed into *speech recognition* module (converting speech signal $\mathbf{X}$ into textual sen-

Amazon/VB Assets
Exhibit 1012
Page 880

tence **W**) and *sentence interpretation* module (parsing sentence **W** into semantic objects **F**). Since the collection of semantic objects **F** is in the linguistic level, it is often referred to as surface semantics.

- **Discourse analysis**—use discourse model $P(S_n \mid \mathbf{F}, S_{n-1})$ to derive new dialog context S_n based on the per-turn semantic parse **F** and the previous context S_{n-1}.
- **Dialog manager**—iterate through the possible actions and pick the most suitable one. The quantitative measures governing operations for dialog manager is called the *behavior model*, $P(\mathbf{A} \mid S_n)$.

The pattern recognition framework can be generalized to multimodal systems as well. For input other than speech signal, you only need to replace the input **X** in the semantic parser with input from an associated modality, e.g., **X** could be input from keyboard typing, mouse clicking, pen input, video, etc. As long as the new semantic parser (replacing speech recognizer and sentence interpretation modules in Figure 1.4) can convert it into appropriate semantic representation, the rest of the system can be identical. Similarly, for different output modality, you just need to replace *message generation* and *text-to-speech* modules with a new rendering mechanism.

In this chapter we first describe the characteristics of spoken languages in comparison with written languages. The structure of dialog is discussed in Section 17.2. Understanding is the most fundamental issue in the field of *artificial intelligence*. The kernel of understanding lies on the representation of semantics (knowledge). Several state-of-the-art semantic representation schemes are discussed in Section 17.3. Based on the architecture of SLU systems illustrated in Chapter 1 (Figure 1.4), major modules are discussed in detail, with the Dr. Who SLU system serving as an example to illustrate important issues.

17.1. WRITTEN VS. SPOKEN LANGUAGES

To construct SLU systems, we need to understand the characteristics of spoken languages. It is worth thinking about possible differences between spoken and written use of language that could be relevant to developing spoken language systems. The following is a typical example of two-agent, task-oriented dialog in action:

Sys: Flight reservation service, how can I help you?
User: One ticket to Honolulu, please
Sys: Anchorage to Honolulu, when would you like to leave?
User: Next Thursday
Sys: Next Tuesday, the 30th of November; and at what time?
User: No, Thursday, December 2nd, late in the evening, and make it first class.
Sys: OK, December 2nd United flight 291, first class. Will you need a car or hotel?
User: No.

Amazon/VB Assets
Exhibit 1012
Page 881

17.1.1. Style

In both spoken and written forms, a communicative setting is established. Both forms involve participants. In the case of written language, we normally expect passivity on the part of the addressee(s), though with e-mail bulletin boards, Web chat rooms, and the like, this assumption can be challenged. The communicative event emerges from personal characteristics of the participants—their mood, goals, and interests. Communication depends both on the actual world knowledge and shared knowledge of the participants and on their *beliefs* about one another's knowledge. Communication can be influenced by the setting in which it takes place, whether in spoken or written mode. Also, different subchannels of supportive communication, such as visual aids, gesture, etc., may be available.

A number of grammatical and stylistic attributes have been found to distinguish conversational from written forms. Biber's analysis [8] distinguishes not only a dimension of modality, but also formality; for example a panel discussion is a relatively formal, yet spoken, modality. Some typical features for which distinctions can be measured include the number of passives, the number of pronouns, the use of contractions, and the use of nominalized forms.[1] An example of the grammatical and stylistic difference continuum that Biber uses is illustrated in Figure 17.1. The variation can be measured along multiple orthogonal scales for different genres. In the SLU case, style can be orthogonal to the modality (dialog or dictation, spoken or written). A crossover case is speech dictation used to create a written document that may never be orally rendered again.

Fortunately, much of the disjuncture between spoken and written forms in grammatical style and lexical choice can be handled by training task-specific and modality-specific language models for the recognizer. For this, only the data need vary, not necessarily the modeling methods. In Figure 17.1, the right-hand side is toward the spoken style, while the

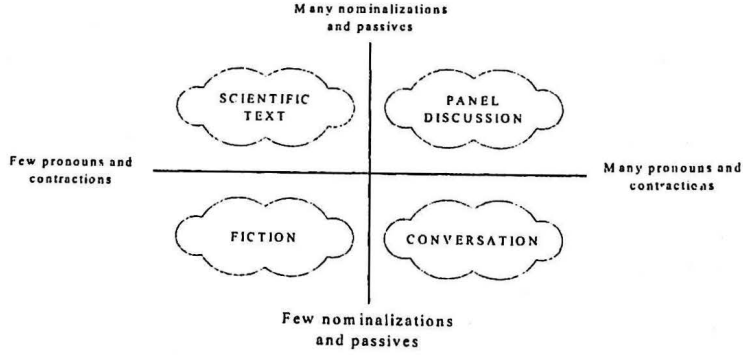


Figure 17.1 Dimensions of written vs. spoken language variation.

[1] Nominalization is a stylistic device whereby a main verb is converted to a noun. For example, *The dean rejected the application unexpectedly* may become: *The rejection of the application by the dean was unexpected.*

Amazon/VB Assets
Exhibit 1012
Page 882

left-hand side is toward the written one. The difference in styles is best illustrated by the fact that the statistical n-gram trained from newspaper text exhibits a very high perplexity when evaluated on conversational Air Travel Information Service (ATIS) texts.

17.1.2. Disfluency

Another issue for spoken language processing is *disfluency*. Spoken dialogs show a large set of problems such as interruptions, corrections, filled pauses, ungrammatical sentences, ellipses, and unconnected phrases. These challenges are unique to spontaneous spoken input and represent a possible further degradation of speech recognizer performance, as current systems often rely on acoustic models trained from read speech, and language models trained on written text corpora. When speech input is used as dictation for document creation, of course, the models would presumably be most appropriate.

There are a number of types of disfluencies in human-human dialog, and, possibly to a lesser extent, in human-computer dialog as well. The more common types of nonlinguistic disfluencies are listed below:

- Filled pauses: *um*
- Repetitions: *the – the*
- Repairs: *on Thursday – on Friday*
- False Starts: I like – what I always get is …

Early work in discourse led to the determination that discourses are divided into *discourse segments*, much as sentences are divided into phrases [18]. In the experiments of [46], CART methods (see Chapter 4) were used to predict occurrence and location of each of the above types of disfluency. A tree was trained from labeled corpora for each type, and the resulting system classified each interword boundary as having no disfluency or one or more of the above types. The feature types used to derive the classification questions included duration of vocalic regions and pauses, fundamental frequency and its derivatives, signal-to-noise ratios, and distance of the boundary from silence pauses. The basic classification task consisted in selecting each of the four disfluency types listed above (D), given the list of prosodic features ($\mathbf{X}$), by computing the maximum of $P(D|\mathbf{X})$. When decision trees were used to supplement the language-model scoring of hypothesis word strings, performance improved.

A number of intriguing regularities were also observed in this work. For example, it was noted that the marked (less common) pronunciation of *the* - */dh iy/* was often used just prior to a production problem, e.g., a disfluent silent pause. Also, it has been noted that the leftmost word of a major phrase or clause is likely to be repeated, as in their example, "*I'll I'll do what I can.*" Continued research on disfluencies may contribute an important secondary knowledge source to supplement text-based language models and *read speech* acoustic models in the future.

Amazon/VB Assets
Exhibit 1012
Page 883

17.1.3. Communicative Prosody

Prosodic attributes of utterances, such as fundamental frequency and timing (cf. Chapter 15), are crucial cues for detecting disfluency. However, prosody can be deliberately manipulated by speakers for deep communicative purposes as well. The speaker may intentionally or subconsciously manipulate the fundamental frequency, timing, and other aspects of voice quality to communicate attitude and emotion. If a conversational interface is equipped to recognize and interpret prosodic effects, these can be taken into account for understanding.

In addition to serving as a disfluency detector, as described above, prosodic analysis modules could aid recognition of:

- Utterance type—declarative, *yes-no* questions, *wh*-question, etc.
- Speech act type—directive, commissive, expressive, representative, declarative, etc. Different speech acts will be described in Section 17.2.2.
- Speaker's attentional state.
- Speaker's attitude toward his/her utterance(s).
- Speaker's attitude to system presentations.
- Speaker's mood or emotional state.

Consider the simple utterance *OK*. This may be used along a range of attitudes and meanings, from *bored contempt*, to *enthusiastic agreement*, to *questioning* and *uncertainty*. The interpretation will depend on both the dialog state context of expectations-to-date and the prosody. Generally, a higher relative F0 in a wider range correlates with submission, involvement, questioning, and uncertainty, while a lower relative F0 in a narrower range correlates with dominance, detachment, assertion, and certainty. Even though acknowledgement words such as *yeah* and *ok* are potentially ambiguous among: true agreement; intention of the listener to initiate a new turn; and simple passive encouragement from listener to speaker, the system may rely on a longer duration and greater pitch excursion of a lexical item such as *yeah* or *ok* to hypothesize genuine agreement with a speaker statement, as opposed to mere acknowledgement.

In addition to correlating with speech acts, F0 and timing can be used to demarcate utterance and turn segments. For example, certain boundary pitch movements and phonemic lengthening systematically signal termination of clauses. In general, a fall to the very bottom of a speaker's range, in a prepausal location, coincides with a clause or sentence boundary. A sharp upturn preceding a significant silence gives an impression of incompletion, perhaps signaling a yes-no question, or may signal an intention by the speaker to carry on with further information, as in the case of list intonation.

The disfluent and prosodic characteristics of the conversational speech are in general very distinct from those of read speech. Thus, we often refer conversational speech as *spontaneous* speech.

Amazon/VB Assets
Exhibit 1012
Page 884

17.2. DIALOG STRUCTURE

The analysis methods discussed in Chapter 11 are focused on single sentences. They are steps along the way, helping to map vague and ambiguous natural language constructions into precise *logical forms* of propositions. In reality, however, the communicative function of language is not a simple, uncomplicated assembly of discrete logical propositions derived from sentences in a one-to-one fashion. In *discourse*, each sentence or utterance contributes to a larger abstract information structure that the user is attempting to construct. Sometimes feedback is directly available to the user or can be inferred. These considerations take us beyond the process of mapping of isolated utterances into logically structured propositions (with simple truth-values).

A set of principles, known collectively as the cooperative principle, is introduced by Grice [9]. It consists of a set of conversational maxims, the violation of which may lead to a breakdown in communication.

GRICE'S MAXIMS

Quantity: speaker tries to be as informative as possible, and gives only as much information as needed
Quality: speaker tries to be truthful, and does not give information that is false or that is not supported by evidence
Relevance: speaker tries to be relevant, and says things that are pertinent to the discussion
Manner: speaker tries to be as clear, as brief, and as orderly as possible, and avoids obscurity and ambiguity

In general, there are five main domains of operation that must be modeled for intelligent conversation systems, although all these areas are linked:

- **Linguistic forms**: all the knowledge a human-computer dialog system requires to perform semantic and syntactic analysis and generation of actual utterances.
- **Intentional state**: goals related to both the task (*Show me all flights ...*), and the dialog process itself (*Please repeat ...*) of the users.
- **Attentional state**: the set of entities at any point in time that can be felicitously discussed and referred to, i.e., the main topic of any stage of interaction.
- **World knowledge**: common sense knowledge and inference. Examples include temporal and spatial concepts and the relation of these to linguistic forms.
- **Task knowledge**: all information relevant to achieving the user's goal in a complete, correct, and efficient fashion.

Amazon/VB Assets
Exhibit 1012
Page 885

Human-computer dialog is multiagent communication. Each agent has to form a notion of the other's beliefs, desires, and knowledge, all of which underlie their intentions, plans, and actions. In a limited application, deep inference may not be possible, and the system may have more or less *hardwired* assumptions about the user, the interaction, and the flow of action. An interaction may be controlled by the system's own rigid schedule of information acquisition. In the research community, such a dialog system—always leading the interaction flow control and not allowing the user to digress—is called *system initiative*. On the other hand, a dialog system is called *user initiative* if it always lets the user decide what to do next. It is often more natural, however, to allow for *mixed initiative* systems, where interaction starts with a user's query or command and the system attempts to derive, via inference or further questioning of the user, all information needed to understand and process a complete transaction. When the user knows clearly what he wants and the system has no trouble catching up, the user is in the driver's seat. However, when the system detects that the user is in a state of confusion, or when it has trouble getting user's intention, the machine will offer guidance or negotiate with the user to steer the dialog back on track.

Whether it is system-initiative, user-initiative, or mixed-initiative, however, the fundamental structure of dialog consists of initiative-response pairs as indicated in Figure 17.2. The *Initiatives* (I) are often issued by users while the *Responses* (R) are issued by the system. As shown in Section 17.2.2, there are many types of Initiatives and Responses and there may also be higher-order structure subsuming a number of I/R pairs in a dialog.

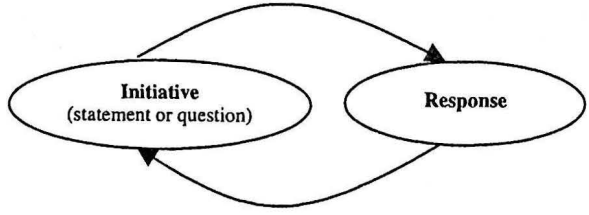


Figure 17.2 The fundamental structure of dialog: initiative and response.

17.2.1. Units of Dialog

The words uttered in a dialog are the surface manifestation of a complex underlying layer of participants' shared interaction knowledge and desires, even when one participant is a computer simulation. It is natural to assume that the *sentence* is a clear and simple chunking unit for dialog, by analogy with written communication. However, since sentences are artificially delimited in written text, researchers in dialog communication usually speak of the *utterance* as the basic unit. An Initiative or Response could consist of one or more utterances. The utterance, however, is not necessarily trivial to define.

It is tempting to posit an equivalence of the notion utterance with *turn*, i.e., an uninterrupted stream of speech from one participant in a dialog. This formulation makes it easy to segment dialog data into utterance units—they are just each speaker's turns. The downside

Amazon/VB Assets
Exhibit 1012
Page 886

is that this kind of *utterance* possibly spans grammatical units that really do have some rough correspondence to traditional sentences (predicate-argument structures), and to which much of the hard-won gains in natural language processing would apply fairly directly. Thus, the use of *turn* as synonymous with utterance unit is probably too broad, though the *turn* may be independently useful for higher-level segmentation.

Turns are building blocks for constructing a common task-oriented understanding among participants. This process is called *grounding*, a set of discourse strategies by which dialog actors (humans in most current research) attempt to achieve a common understanding, and come to feel confident of the other participants' understanding. In other words, conversational partners are finding or establishing *common ground.*

Turns may have their own typology. For example, a *speaking turn* conveys new information, while a *back-channel turn* is limited to acknowledgement or encouragement, such as *OK, really?*, etc. The turns themselves consist of linguistic substructures, such as sentences, clauses, and phrases. If we assume that turns can be segmented, by grammatical and/or prosodic criteria, into utterances, we can then begin to explore distinct types of utterances, their properties, and their communicative functions.

Finally, dialogs are not flat streams of unrelated turns or utterances. The utterances that make up a dialog have higher-order affiliations with one another. A discourse segment would thus consist of groups of related utterances organized around a common dialog subtask, perhaps spanning turns.

17.2.2. Dialog (Speech) Acts

In simpler applications, the amount and sophistication of world knowledge can be kept to a minimum, and attentional state can be modeled simply as the complete set of task-specific entities. A layer of structure has therefore been sought to link linguistic forms with task knowledge or operations in a theoretically appropriate fashion, which also yields an implicit understanding of intentional state. This is necessary because the function of utterances in discourse cannot be predicted strictly on the basis of their surface grammatical form. The layer of structure that can abstract away from linguistic details and can map well to formulation of goals is called *dialog acts* [42]. Dialog acts are also often referred to as *speech acts* that group infinite families of surface utterances into abstract functional classes. They are traditionally classified into five broad categories:

- **Directive**: The speaker wants the listener to do something.
- **Commissive**: The speaker indicates that he or she will do something in the future.
- **Expressive**: The speaker expresses his or her feelings or emotional response.
- **Representative**: The speaker expresses his or her belief about the truth of a proposition.
- **Declarative**: Speaker's utterance causes a change in external, nonlinguistic situation.

Amazon/VB Assets
Exhibit 1012
Page 887

Table 17.1 A simple dialog analyzed with dialog acts.

Utterance	Form	Function
Do you have the butter?	Y/N-question	REQUEST-ACT
Sure. (passes butter)	statement	COMMIT-TO-ACTION-ACT

While this analysis is somewhat coarse, speech act theory has influenced all current work on human-computer dialog, except the very simplest and most rigid systems. Because dialog functions can be realized with a bewildering variety of linguistic forms, researchers have posited systems of functional abstractions. Speech acts are functional abstractions over variation in utterance form and content. Declare, request, accept, contradict, withdraw, acknowledge, confirm, and assert are all examples of speech acts—things we are attempting to do with speech. An example of dialog acts and their relation to syntactic form is shown in the two-turn dialog in Table 17.1.

The relation between speech acts and linguistic forms (utterances) is a many-to-many mapping. That is, a single linguistic form, such as *OK*, could realize a large number of speech acts, such as *request for acknowledgment* or *confirm*, etc. Likewise, a single speech act, such as *agreement*, could be realized by a variety of linguistic forms, such as *ok, yes, you bet*, etc. In a particular application, special task-specific speech acts may be used to supplement the universal inventory.

Tagging of dialog utterance data with speech-act labels can add useful information for training models. There are a number of ways that dialog-act analysis could be useful:

- *Speech recognition*: Given a history, we can predict the most likely dialog act type for the *next* utterance, so that specialized language models may be applied.
- *Spoken language understanding*: Given a history, and a transcription/parse of the current utterance, we can identify the user's intentions, so that the system can respond appropriately.
- *Semantic authoring*: It is tedious for each team designing or customizing a new application area for SLU to have to wrack their brains for all the ways a given generic function, such as *request* or *confirm*, might be realized linguistically. Libraries of speech acts (form-to-function mappings) may reduce the work in new-domain adaptation of systems.

An example of a practical dialog tagging system that could be the foundation of speech-act analysis is the *Dialog Act Markup in Several Layers* (DAMSL) system [14], which has been used and adapted for a variety of projects. This is a system for annotating dialog transcriptions with speech-act labels and corresponding structural elements. The structuring is based on a loose hierarchy of: discourse segment, turn, utterance, and speech act. The tags applied to utterances fall into three basic categories:

Amazon/VB Assets
Exhibit 1012
Page 888

- **Communicative Status**: records whether the utterance is intelligible and whether it was successfully completed. It is mainly used to flag problematic utterances that should be used for data modeling only with caution—Uninterpretable, Abandoned, or Self-talk. *Uninterpretable* is self-explanatory. *Abandoned* marks utterances that were broken off without, crucially, adding any information to the dialog. *Self-talk* is a note that, while an utterance may contain useful information, it did not appear to be intentionally communicated. Self-talk can be considered reliable only when the annotator is working from speech data.
- **Information Level**: a characterization of the semantic content of the utterance. This is used to specify the kind of information the utterance mainly conveys. It includes *Task* (Doing the task), *Task-management* (Talking about the task), *Communication-management* (Maintaining the communication), and *Other-level*. Task utterances relate directly to the business of the transaction and move it toward completion. Task-management utterances ask or tell about the task, explain it perhaps, but do not materially move it forward. Communication-management utterances are about the dialog process and capabilities. The Other level is for unclear cases.
- **The Forward/Backward Looking Function**: how the current utterance constrains the future/previous beliefs and actions of the participants and affects the discourse. Forward Looking functions introduce new information or otherwise move the dialog or task completion forward, while Backward Looking Functions are tied to an antecedent, a prior utterance which they respond to or complete. This distinction is the DAMSL reflection of the common observation that dialogs have a tendency to consist of Initiation/Response pairs. The core of the system is the set of particular act types. The core Forward/backward Looking tags are listed in Table 17.2 and Table 17.3.

Table 17.2 Forward looking tags.

Forward Looking Tags	Example
assert	I always fly first class.
reassert	No, as I said, I always fly first class.
action-directive	Book me a flight to Chicago.
open-option	There's a red-eye flight tonight ...
info-request	What time is it?, Tell me the time.
offer	I can meet at 3 if you're free.
commit	I'll come to your party.
conventional opening	May I help you?
conventional closing	Goodbye.
explicit-performative	Thank you, I apologize.
exclamation	Ouch! Darn!

Amazon/VB Assets
Exhibit 1012
Page 889

Table 17.3 Backward looking tags.

Backward Looking Tags	Example
accept	(Will you come?) Yes. [and/or, I'll be there at 10.]
accept-part	(Will you come with your wife?) I'll come, she may be busy.
reject	(Will you come?) No.
reject-part	(Want fries and a shake with that burger?) Just the hamburger and fries, please.
maybe	Maybe.
signal-nonunderstanding	What did you say?
acknowledgment	OK.
answer	(Can I fly nonstop from Anchorage to Kabul?) No.

Multiple tags may appear on any given utterance. In the example shown in Figure 17.3, B's utterance is coded as opening the option of buying (from B), asserting the existence of the sofas, and functioning as an offer or solicitation.

Action-directive	`A: Let's buy the living room furniture first.`
Open-option/Assert/Offer	`B: OK, I have a red sofa for $150 or a blue one for $200`

Figure 17.3 A tagged dialog fragment.

The DAMSL system is actually more complex than the example demonstrated above, since subsets of the tags are grouped into mutually exclusive options for a given general speech function. For example, there is a general *Agreement* function, under which the *accept*, *accept-part*, *reject*, and *reject-part* tags are grouped as mutually exclusive options. Above the level of those groupings, however, a single utterance can receive multiple nonexclusive tags. For example, as illustrated in Figure 17.4, the assistant may respond with a countersuggestion (a kind of action-directive) that rejects part of the original command.

Action-directive	`utt1 oper: Take the train to Avon via Bath`
Action-directive/Reject-part(utt1)	`utt2 asst: Go via Corning instead.`

Figure 17.4 A tagged dialog fragment, showing that utterances can be tagged with multiple nonexclusive tags.

Amazon/VB Assets
Exhibit 1012
Page 890

The prototypical dialog turn unit in simple applications would be the I/R pair *info-request/answer*, as in the interaction shown in Figure 17.5 between an operator (planner) and an assistant regarding railroad transport scheduling [1].

The example in Figure 17.5 illustrates a dialog for a railway-scheduling task. The turns are numbered T1–T4, the utterances within turns are also numbered sequentially, and the speaker identity alternates between oper: and asst:. The tagging is incomplete, because, for example, within the *ansl* sequence, each utterance is performing a function, asserting, acknowledging, etc. The example in Figure 17.6 is a more completely annotated fragment, omitting turn numbers.

```
info-req     T1 utt1 oper:    where are the engines?
ansl         T2 utt2 asst:    there's an engine at Avon
  |          T3 utt3 oper:    okay
  |          T4 utt4 asst:    and we need
               utt5 asst:    I mean there's another in Corning
```

Figure 17.5 A tagged dialog fragment in railroad transport scheduling.

```
info-req/assert        utt1 oper:  and it's gonna take us also an
                                   hour to load boxcars right
ans/accept(utt1)       utt2 asst:  right
assert                 utt3 oper:  and it's gonna take us also an
                                   hour to load boxcars
accept(utt1)           utt4 asst:  right
```

Figure 17.6 A tagged dialog fragment, showing backward-looking utterances.

The example in Figure 17.6 shows backward-looking utterances, where the relevant antecedent in the dialog is shown (in parentheses) as part of the dialog coding.

More elaborate variants of DAMSL have been developed that extend the basic system presented here. Consider, for example, the SWITCHBOARD Shallow-Discourse-Function Annotation SWBD-DAMSL [27]. This project used a shallow discourse tag set of 42 basic tags (frequent composed tags from the large set of possible multitags) to tag 1155 5-minute conversations, comprising 205,000 utterances and 1.4 million words, from the SWITCHBOARD corpus of telephone conversations. Distributed by the Linguistic Data Consortium[2] [28], SWITCHBOARD is a corpus of spontaneous conversations that addresses the growing need for large multispeaker databases of telephone bandwidth speech. The corpus contains 2430 conversations averaging 6 minutes in length—in other words, over 240 hours of recorded speech, and about 3 million words of text, spoken by over 500 speakers of both genders from every major dialect of American English.

More detailed tags are added to DAMSL to create SWBD-DAMSL, most of which are elaborations of existing DAMSL broad categories. For example, where DAMSL has the simple category *answer*, SWBD-DAMSL has: *yes answer*, *no answer*, *affirmative non-yes*

[2] http://www.ldc.upenn.edu

Amazon/VB Assets
Exhibit 1012
Page 891

answer, *negative non-no answers*, *other answers*, *no plus expansion*, *yes plus expansion*, *statement expanding y/n answer*, *expansions of y/n answers*, and *dispreferred answer*. SWBD-DAMSL is intended for the annotation and learning of structure in human-human dialog, and could be considered overkill as a basis for describing or constructing grammars for most limited-domain human-computer interactions of today. But the more sophisticated agent-based services of the future will need to assume ever-greater linguistic sophistication along these lines.

One fact noted by the SWBD-DAMSL researchers, which may not apply directly to task-directed human-computer interactions but which casts interesting light on human communication patterns, is that out of 1115 conversations studied, simple nonopinion statements and brief acknowledgements together constituted 55% of the conversational material! If statements of opinion (including simple stuff like *I think it's great!*), expressions of agreement (*That's right!*), turn breakoffs and no-content utterances (*So...*), and appreciative acknowledgements (*I can imagine.*) are added to this base, 80% of the utterances are accounted for. This relative poverty of types may bode well for future attempts to annotate and predict utterance function automatically. The DAMSL scheme is challenging to apply automatically, because it relies on complex linguistic and pragmatic judgments of the trained annotators.

17.2.3. Dialog Control

The system's view of how the dialog should proceed is embodied in its management strategy. Strategy is closely connected to the concept of *initiative* in dialog, meaning basically who is controlling the interaction. Different dialog initiatives are defined in Section 17.2. Initiative can be seen as a continuum from system controlled to user controlled. As background for the dialog management discussion, some important steps along this continuum can be identified:

- *System directs*—The system retains complete dialog control throughout the interaction. The system chooses the content and sequence of all subgoals and initiates any dialog necessary to obtain completion of information from the user for each transaction. This style is often referred as *system initiative*.
- *System guides*—The system may initiate dialog and may maintain a general plan, but the sequence of information acquisition from the user may be flexible, and system subgoals and plans may be modified in response to the user's input. This style is often referred as *mixed initiative*.
- *System inform*—The user directs the dialog and the system responds as helpfully as possible, which may include presentation of relevant data not specifically requested by the user but which the system believes could be helpful. This style also belongs to *mixed initiative*, though users control most of the dialog flows.
- *System accepts*—This is the typical human-computer interaction in traditional systems (whether it is a GUI-, command-line-, or natural language-based sys-

Amazon/VB Assets
Exhibit 1012
Page 892

tem). The system interprets each command without any attempted inference of a deeper user plan, or recommendation of any suitable course of action. This style is referred as *user initiative.*

17.3. SEMANTIC REPRESENTATION

Most SLU systems require an internal representation of *meaning* that lends itself to computer processing. In other words, we need a way of representing semantic entities, which are used at every possible step. In general, an SLU system needs to deal with two types of semantic entities. The first type is *physical entities*, which correspond to the real-world entities. Such representation is often referred as knowledge representation in the field of artificial intelligence. The second type is *functional entities*, which correspond to a way of unambiguously representing the meaning or structure of situations, events, and concepts that can be expressed in natural language. Such representations are often similar to the *logical form* introduced in Chapter 2. Processing may include operations such as determining similarity or identity of events or entities, inference from a state of affairs to its logical consequences, and so on. Here, we briefly review some general properties of the common semantic representation frameworks.

17.3.1. Semantic Frames

Semantic objects are used to represent real world entities. Here, we assume that the domain knowledge conforms to a relational or objected-oriented database, of which the schema is clearly defined. We use the term *entity* to refer to a data item in the domain (a row in a database table), or a function (command or query) that can be fulfilled in the domain. A column in the database table is called an entity attribute, and each database table is given an entity type. Through a small subset of its attributes, an entity can be realized linguistically in many fashions. We call each of them a *semantic class*. For example, a person can be referred to in terms of her full name (*Angela*), a pronoun anaphora (*her*), or her relationships to others (*Christina's manager*). In this case, one can derive three semantic classes for the entity type.

Semantic classes can be viewed as a type definition to denote the objects and describe the relations that hold among them. One of the most popular representations for semantic classes is the *semantic frame* [31]—a type of representation in which a semantic class (concept) is defined by an entity and relations represented by a number of attributes (or slots) with certain values (the attributes are filled in for each instance). Thus, frames are also known as *slot-and-filler* structures.

We could, for example, define a generalized frame for the concept *dog*, with attributes that must be filled in for each particular instance of a particular dog. A type definition for the concept dog appears in Figure 17.7. Many different notational systems have been used for frames [51]. For these introductory examples, we use a simple declarative notation that should be fairly intuitive.

Amazon/VB Assets
Exhibit 1012
Page 893

```
[DOG:]-
  [SUPERTYPE]->[mammal]
  [NAME]->()
  [BREED]->()
  [LOC]->()
  [Color]->()
```

Figure 17.7 A semantic frame representation for *dog*.

When we need to describe a particular dog, say *Lassie*, we create an *instance definition*, as shown in Figure 17.8. The knowledge base supporting a typical dialog system consists of a set of type definitions, perhaps arranged in an inheritance hierarchy, and a set of instances.

```
[DOG:]-
   [NAME]->(Lassie)
   [BREED]->(Collie)
   [LOC]->()
   [Color]-()
```

Figure 17.8 An instance of semantic frame *dog*.

Fillers in semantic frames can be attained by attachment of inheritance, procedures or default. Attributes in frame can typically be inherited, as the Lassie instance inherits mammalian properties from the DOG type definition. In some cases, properties of a particular dog may be dynamic. Sometimes *attached procedures* are used to fill dynamic slots. For example, the location of a dog may be variable, and if the dog has a *Global Positioning System* (GPS) chip in its collar, the LOC property could be continually updated by reference to the GPS calculations. Furthermore, procedures of the type `when-needed` or `when-filled` can also be attached to slots. Finally, some default value could provide a typical value for a slot when the information for that slot is not yet available. For example, it might be appropriate to set the default color for *dog* frame as white when such information is not available. For frames without a default-value slot, it is natural to define *mandatory* slots (slots' values must be filled) and optional slots (slots could have null value). For the *dog* frame, it is reasonable to assume the NAME slot should be mandatory while the COLOR slot can be optional.

Often *descriptions* can be attached to slots to establish constraints within or between frames. Description may have connectives, co-referential (description attached to a slot are attached to another) and declarative conditions. For example, the *return-date* slot of a *round-trip* itinerary frame must be no earlier than the *departure-date* slot, and this constraint can be specified by descriptions in both slots. Descriptions can also be inherited and are often implemented by a special procedure (different from the slot-filling procedure) attached to the slot.

The main motivation for having multiple semantic classes for each entity type is to better encapsulate the language, semantic, and behavior models based on the domain knowledge. While the entity relationships capture the domain knowledge, the semantic class hier-

Amazon/VB Assets
Exhibit 1012
Page 894

archy represents how knowledge can be expressed in the semantics of a language and thus can cover linguistic variation. The concept of semantic objects/classes is similar to that of objects/classes in modern *object-oriented programming*. The semantic classes in Dr. Who [59, 60] are a good illustration of borrowing some important concepts from object-oriented programming to enhance the effectiveness and efficiency of using semantic objects/classes to represent domain knowledge and linguistic expressions.

The semantic grammar used in the Dr. Who Project [58] contains the definitions of semantic classes that refer to real-world or functional entities. A semantic class is defined as a semantic frame containing a set of slots that need to be filled with terminal (verbatim) words or with recursive semantic class objects. For example, *ByRel* is a semantic class that has the type PERSON. The semantic grammar specifies that it has two slots—one has to be filled with an object of a semantic class having the type PERSON, and the other has to be filled with an object of a semantic class having the type P_RELATION. On the other hand, the syntax grammar for this semantic class is specified by the <cfg> tags. Within <cfg> tags, several production rules can be specified to provide linguistic constraints (orders) of possible expressions for this semantic class. The syntactic aspect of semantic classes will be described further in Section 17.4.1.

17.3.1.1. Type Abstraction

As described above, a physical *entity* is an element in the real world that an application has to deal with and wishes to expose to the user via natural language. Since a physical entity can be referred to in many different ways, different semantic classes may have the same type. In Figure 17.9, a person can be referred to in terms of his name (*Peter*) or his relation to another person (*Peter's manager*); therefore, both semantic classes ByName and ByRel can share the same type, PERSON.

Semantic classes are designed to separate the essential attributes of a semantic object from its physical realizations. A semantic class may refer to an entity, and the entity is called the type of the semantic class. The attributes of a semantic class can, in turn, be semantic classes themselves. The concept behind semantic classes is identical to the mechanism known as type abstraction commonly employed in software engineering using a strongly typed programming language. Semantic class can be recursive, as demonstrated in Figure 17.9; a ByRel semantic class of type PERSON contains an attribute of PERSON type. Since the entities can be *nested*, i.e., a database column can in turn refer to another table, an attribute in the semantic class can also be an entity type. From an understanding point of view, a semantic class is an abstraction of the collection of semantic objects that have the same attributes and usually can be expressed, and hence be understood, in similar manners. Under this view, a semantic object is just an instantiation.

Another argument for type abstraction is that the multitude of semantic objects is usually a result of the numerous ways and perspectives that can be used to describe a physical entity. Quite often in an understanding system it is more important to correctly identify the entity of interest than to capture the mechanism that describes it. For instance, one may refer to a person by his name, job function, relations to others, or, in a multimodal environment,

Amazon/VB Assets
Exhibit 1012
Page 895

by pointing to his photo on a display. All these references lead to semantic objects that are apparently distinct yet should be associated with the same physical entity. Accordingly, it is often useful to segregate the conceptual manifestation and its realizations into different layers of abstraction so that the semantic objects can be better organized and managed. Type abstraction allows the discourse sentence interpretation module to perform robust parsing, since sentence fragments can be parsed into its semantic class type that can be filled into slots with the same correspondent semantic type, as discussed in Section 17.4.1.

```
<!-- semantic class definition for ByRel that has type PER-
SON -->
<class type="PERSON" name="ByRel">
    <slot type="PERSON" name="person"/>
    <slot type="P_RELATION" name="p_relation"/>
    <cfg>
        <prod> [person] [p_relation] </prod>
        <prod> [p_relation] of [person] </prod>
    </cfg>
</class>
<!-- semantic class definition for ByName that has type PERSON too -->
<class type="PERSON" name="ByName">
    <slot type="FIRSTNAME" name="firstname"/>
    <slot type="LASTNAME" name="lastname"/>
    <cfg>
        <prod> [firstname] [lastname] </prod>
        <prod> [firstname] </prod>
        <prod> [lastname] </prod>
    </cfg>
</class>
<!-- semantic class definition for FIRSTNAME and LASTNAME -->
<verbatim type="FIRSTNAME"
    <cfg>
        <prod> john | john's | peter ... </prod>
    </cfg>
</verbatim >
<verbatim type="FIRSTNAME"
    <cfg>
        <prod> smith | smith's | shaw ... </prod>
    </cfg>
</verbatim >
<!-- semantic class definition for P_RELATION -->
<verbatim type="P_RELATION"
    <cfg>
        <prod> manager | father | mother | ... </prod>
    </cfg>
</verbatim >
```

Figure 17.9 The semantic classes of type PERSON as implemented in Dr. Who.

Amazon/VB Assets
Exhibit 1012
Page 896

Finally, type abstraction provides a unified framework for resolving *relative expressions* in the discourse analysis module. Type matching often serves to impose strong constraints between real-world entities and relative expressions. The resolution of relative expressions is discussed in Section 17.5.

17.3.1.2. Property Inheritance

Introducing inheritance into the semantic class hierarchy further augments the multilayer abstraction mentioned above. Class *A* is said to inherit or be derived from class *B* if class *A* possesses all the attributes of class *B*. In this case, class *A* is called the derived class and class *B* the base class. Inheritance is a mechanism to propagate knowledge and properties through the structural relationships of semantic classes. It is crucial for many types of intelligent behavior, such as deducing presumed facts from general knowledge and assuming default values in lieu of explicit and specific facts.

Perhaps the strongest motivation to employ inheritance is to facilitate the multilayer abstraction mentioned above. Very often, a base class is constructed with the general properties of a type of semantic objects, and a collection of more specific classes are derived from the base class to support the various embodiments of the underlying type of the semantic objects. For example, a semantic class hierarchy for the reference to a person can have the methods (e.g., by name, job function) and the media (e.g., speech, handwriting) of reference as the first layer of derived classes. One can then cross-match the viable means (e.g., by name via speech, by name via handwriting) and develop the second layer of derived classes for use in the real applications.

17.3.1.3. Functionality Encapsulation

The goal of abstraction is to reduce the complexity in describing the world—in this case, the semantic objects and their relations. One can inspect the quality of abstraction by examining the extent to which the constructs, i.e., semantic classes, are self-contained, and how proliferating they have to become in order to account for novel scenarios. Studies in data structure and software engineering propose the notion of encapsulation, which suggest that individual attributes have local rather than global impacts. This principle also serves as a guideline in designing the semantic class.

Semantic class encapsulation can be elaborated in two aspects: syntactic and semantic. The syntactic encapsulation refers to the constraint that each attribute in a semantic class can only have relations to others from the same class. The collection for these relations is called the *semantic grammar,* which specifies how a semantic object of this type can be identified. In Figure 17.9, the tag <CFG> specifies how the semantic class can be referred to syntactically via a context-free grammar (CFG). For the class ByRel, the specified syntax indicates that expressions like *Peter's manager* and *manager of Peter* are legal references to semantic class ByName. The semantic encapsulation, on the other hand, dictates the actions and the discourse context under which they may be taken by a semantic class. This is discussed further in Section 17.5.

Amazon/VB Assets
Exhibit 1012
Page 897

As described in 17.2.2, it is a nontrivial task to determine the types of speech acts. The semantic frame is an abstraction of the speech acts, the domain knowledge, and sometimes even the application logic. Once we have this rich semantic representation, how to parse spoken utterances into the semantic frames becomes the critical task. Nonetheless, the combination of semantic frames and the semantic parser alleviates the need for a dedicated module for determining speech acts.

Semantic frames and associated robust parsing (described in Section 17.4.1) have been widely used in spoken language understanding. For detailed description of semantic classes and frames, you can refer to [58, 65].

17.3.2. Conceptual Graphs

The semantic-representation requirement has led to development of a proposal to standardize the logical form that may form the basis of the internal semantics and semantic interchange of natural language systems, including dialog processing, information retrieval, and machine translation. The proposal is based on conceptual graphs derived from Charles Sanders Peirce [38] and the various types of semantic networks used in artificial intelligence research.

The *conceptual graph* (CG) proposal [53] specifies the syntax and semantics of conceptual graphs as well as formats for graphical and character-based representation and machine-based exchange. In the terms of the proposed standard, a conceptual graph (CG) is an abstract representation for logic with nodes called *concepts* and *conceptual relations*, linked together by arcs. In the graphical representation of a CG, concepts are represented by rectangles, and conceptual relations are represented by circles or ovals. The ordinary phrasing for the association of relations (circles) to concepts (rectangles) is *has a(n)* for arrows pointing toward the circle and *is a(n)* for arrows pointing away.

Figure 17.11 illustrates a conceptual graph for the sentence *Eric is flying to Boston by airplane*. The mnemonic meaning of the arrows is: *Fly* has an agent who is a person, *Eric*, and a destination *Boston*. The proposal also specifies a linear form, as shown in Figure 17.10. In the form, concepts are in square brackets and conceptual relations are in parentheses. The hyphen means that relations of a given concept continue on subsequent lines, as shown in Figure 17.11.

```
[Fly] -
     (Agent)->[Person: Eric]
      (Dest)->[City: Boston]
      (Inst)->[Airplane]
```

Figure 17.10 A linear form representation of *Fly* has an agent who is a person, *Eric*, and a destination *Boston*.

Amazon/VB Assets
Exhibit 1012
Page 898

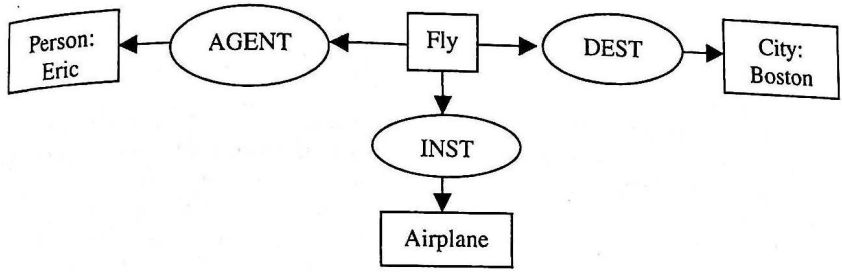


Figure 17.11 CG display form for *Eric is flying to Boston by airplane* [53].

Each concept has a type and a (possibly empty) referent. An empty referent means that at least one, unspecified example of the type is assumed to exist somewhere (an existential quantifier). So, in Figure 17.10, the type is present, but the referent is left unspecified. In an application, the referent can be completed by referring to a train-schedule database and inserting a particular instance of a scheduled train departure time, location, and number. The *valence* of a relation is the number of required concepts that it links. For example, as shown in Figure 17.12, the relation *between* would be a conceptual relation of valence 3, because typically (something/somebody) is *between* one (something/somebody) and another (something/somebody), as in the familiar English idiom "*somebody is between a rock and a hard place*" (meaning, *to be in great difficulty*). This corresponds to the linear form, as shown in Figure 17.13.

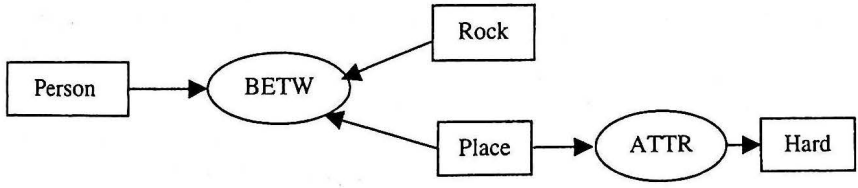


Figure 17.12 CG display form for *a person is between a rock and a hard place* [53].

```
[Person]<-(Betw) -
        <-1-[Rock]
        <-2-[Place]->(Attr)->[Hard]
```

Figure 17.13 A linear form representation of *A person is between a rock and a hard place.*

17.4. SENTENCE INTERPRETATION

We follow the convention of most modern SLU systems—treating semantic parser as a two-step pattern recognition problem (speech recognition followed by sentence interpretation). This convention has the advantage of modular design of SLU systems. Thus, the same SLU

Amazon/VB Assets
Exhibit 1012
Page 899

system can be used for text input. However, a unified semantic parser [50, 62] may achieve better accuracy, because no hard decision needs to be made before picking the optimal semantic representation.

The heart and soul of the *sentence interpretation* module is how to convert (translate) a user's query (sentence) into the semantic representation. In other words, one has to fill the semantic slots with information derived from the content (words) in the sentence. In this section we describe two popular approaches to accomplish this task. Although they can be perceived as pattern matching methods, they differ in the matching mechanism.

17.4.1. Robust Parsing

Due to the nested nature of semantic classes, the semantic representation **F** in Eq. (17.1) can itself be a tree of semantic objects. A user's utterance may consist of disjoint fragments that may make sense at the discourse level. For instance, in the context of setting up a meeting, the utterance "*Peter Duke at a quarter to two*" can be parsed into two semantic objects: a person and the meeting time. Therefore, the sentence interpretation module must deal with sentence fragments.

The analysis of spoken language is a more challenging task than the analysis of written text. The major issues that come to play in parsing spontaneous speech are speech disfluencies, the looser notion of grammaticality that is characteristic of spoken language, and the lack of clearly marked sentence boundaries. The contamination of the input with errors of a speech recognizer can further exacerbate these problems. Most natural language parsing algorithms are designed to analyze grammatical input. These algorithms are designed to detect ungrammatical input at the earliest possible opportunity and to reject any input that is found to be ungrammatical in even the slightest way. This property, which requires the parser to make a complete and absolute distinction between grammatical and ungrammatical input, makes such formal parsers fragile for spontaneous speech, where completely grammatical input is the exception more than the rule. This is why a robust parser is needed.

In Chapter 11, context-free grammars (CFG) can be written to analyze the structure of entire sentences. It is natural to extend CFG as a pattern matching vehicle. For example, a question such as "*Where would you like to go*?" might be used to solicit a response from a user, who might respond, "*I would like to fly to Boston.*" The following grammar might be used to parse the response:

```
S → NP VP
NP → N
VP → VCluster PP
VCluster → would like to V
V → go | fly
PP → prep NP
N → Boston | I
Prep → to
```

Amazon/VB Assets
Exhibit 1012
Page 900

The resulting phrase structure, characterizing the entire sentence, would be:

```
[S [NP [N I ] ] [VP [VCluster would like to [V fly ] ] [PP
[prep to ] [NP [N Boston]]]]]]
```

This structure in turn can provide the foundation for subsequent semantic analysis. Thus, the grammar is adequate for the example response and can be easily extended to cover more city names by expanding the **N** → rule, i.e., by enlarging the lexicon. It has some deficiencies, however. Some of the problems are purely formal or logical in nature, such as the fact that "*Boston would like to go to I*" can be equally parsed. These flaws can be addressed with formal fixes (e.g., a more refined category system), but, in any case, they are not crucial for the practical system designer, because pathological examples are rare in real life. The deeper problem is how to deal with legitimate, natural variations.

The user might respond with any of the following:

```
To Boston
I'm going to Boston.
Well, I want to start in New York and get to Boston by the
day after tomorrow.
I'm in a big hurry; I've got a meeting in Boston.
OK, um, wait a second... OK, I think I've gotta head for Bos-
ton.
```

The above sentences incorporate different kinds of variation for which a *sentence coverage* grammar typically has trouble accounting. For this reason, dialog system designers have gravitated to the idea of *robust parsing*. Robust parsing is the idea of extracting all and only the usable chunks of simple meaning from an utterance, ignoring the rest or treating it as noise or filler. Small grammars can be written that scan a word lattice (see Chapter 13) or a word sequence for just those particular items in which they specialize. For example, a *Destination* grammar, not intended to span an entire utterance, can skim each of the complex utterances above and find the Destination in each case:

```
Destination → Preposition CityName
Preposition → to | for | in
CityName → Boston | ...
```

The noise or *filler* elements might include nonspeech noise (cough, laugh, breath, hesitation), elements of *phatic* communication (greetings, polite constructions), irrelevant comments, unnecessary detail, etc. As a user becomes accustomed to the limited yet practical domain of a system's operations, it is expected that variant phrasings would diminish, since they take longer to utter and contribute very little, though disfluencies would always be an issue.

Amazon/VB Assets
Exhibit 1012
Page 901

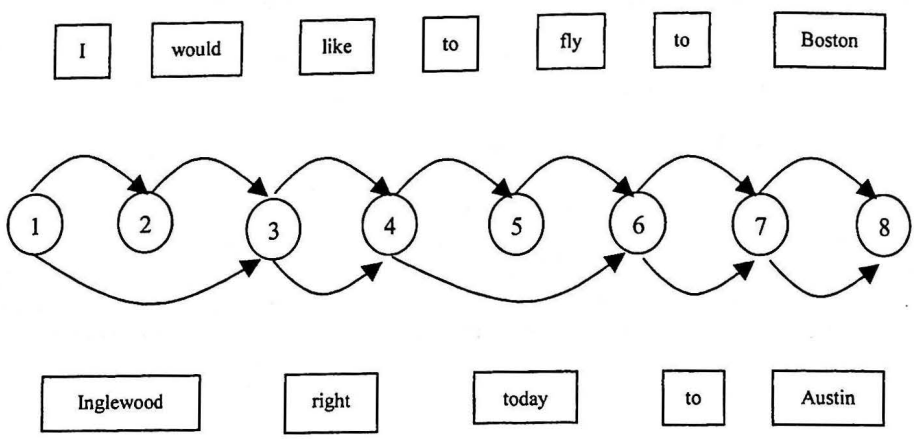


Figure 17.14 Word graph for hypotheses [61].

The original word graph or lattice from the speech recognizer might consist of nodes, representing points in time, and edges representing word hypotheses and acoustic scores for a given span in the utterance. Figure 17.14 illustrates a sample of word graph for the example "*I would like to fly to Boston*" with competing hypotheses. Using the *Destination* grammar on the word graph in Figure 17.14 will skip the earlier parts of the possible sentence hypotheses and identify the short fragment from node 6 to node 8 as a destination. If only the *Destination* grammar were active, a new view of the word graph would result in Figure 17.15.

This example shows that potential and legitimate ambiguities can persist even with flexible grammars of this type, but the key potential meanings have been identified. A robust parser that is capable of handling the example needs to solve the following three problems:

- *Chunking*: appropriate segmentation of text into syntactically meaningful units;
- *Disambiguation:* selecting the unique semantically and pragmatically correct analysis from the potentially large number of syntactically legitimate ones returned; and
- *Undergeneration:* dealing with cases of input outside the system's lexical or syntactic coverage.

Grammars developed for spontaneous speech should concentrate on describing the structure of the meaningful clauses and sentences that are embedded in the spoken utterance. The goal of the parser is to facilitate the extraction of these meaningful clauses from the utterance, while disregarding the surrounding disfluencies. We use the semantic grammar in the Dr. Who SLU engine [61] to illustrate how this works.

Amazon/VB Assets
Exhibit 1012
Page 902

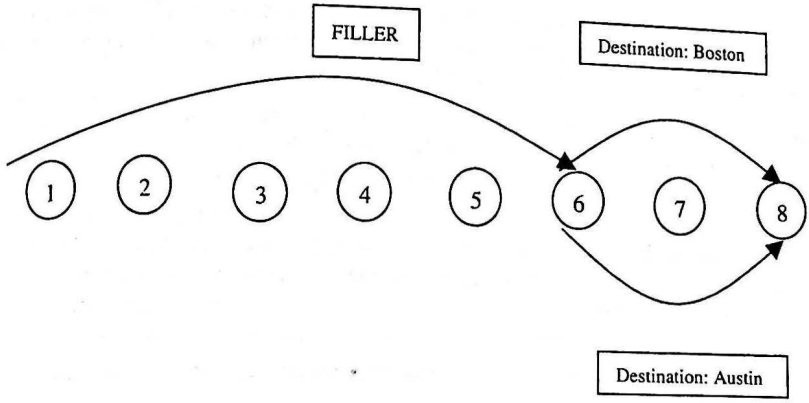


Figure 17.15 Word graph for hypotheses if only the *Destination* grammar is active [61].

As shown in Figure 17.9, a Dr. Who semantic class mostly contains a set of slots that need to be filled with terminal words (verbatim) or with recursive nonterminal semantic classes. Strictly speaking, this semantic class grammar can hardly be called a grammar, since it is primarily used to define the conceptual relations among Dr. Who entities rather than the language expressions that are used to refer to the entities. The syntactic expression is specified by optional CFGs associated with each semantic class. In general, the syntactic grammars need to deal with three kinds of variation in surface linguistic form:

1. *Variation within a slot*—When CFG is missing in the definition of semantic classes, the grammar could allow flexible assembly of an expression. For example, if the <cfg> tags in Figure 17.9 are omitted, any sequence that contains a word of a P_RELATION typed class and a word of a PERSON typed class can be an expression referring to a semantic object of *ByRel* such as *John's father, father of John,* or even *John loves his father.* Thus, CFGs are often specified within the semantic slot to provide linguistic constraints without over-generating.
2. *Variation in the order of frame presentation*—Many systems [64, 66] employ an island-driven robust parsing strategy where the slots in the semantic frames are filled by language fragments from parsing. Parsing of the slots is order independent. Thus utterances "*Schedule a meeting with John at 3 PM*" and "*Schedule a meeting at 3 PM with John*" can be processed without problems.
3. *Disfluencies and irrelevancies*—Disfluencies and irrelevancies are unavoidable for spoken language input. The system has to deal with real utterances such as "*I'd really like to know whether a meeting by 3 PM would be at all possible for John.*"

Amazon/VB Assets
Exhibit 1012
Page 903

To cope with these requirements, the robust parsing algorithm [61] is typically implemented as an extension of the bottom-up chart-parsing algorithm discussed in Chapter 11. There are a number of additional requirements for robust parsing:

- The requirement that a hypothesis and a partial parse have to cover adjacent words in the input is relaxed here. This effectively skips the words and enables the parser to omit unwanted words in input sentences.
- The combination of a hypothesis with a new partial parse taken from *agenda* results in multiple new hypotheses. Those hypotheses may have different critical position number. In other words, they are expecting different partial parses. This effectively skips the symbols in a rule, so the parser can continue its operation even if something expected by the grammar does not exist.
- The sequential order in which the partial parses are taken out from the agenda is crucial here. A partial parse that has the minimum span and highest score and that covers the word closest to the sentence start position (in that order) has the highest priority.

In a robust parser, if there is already a parse g that has the same symbol and span as the new parse h, we need to compare their scores so we only keep the better one. The parse scoring can be the likelihood of the parse with respect to a heuristic CFG enhanced with a mechanism of assigning probability for insertions and deletions. It can also be based on heuristics when no training data is available. The typical heuristic values may include the number of words covered by a parse; the number of rule symbols skipped in the parse tree; the number of nodes in the parse tree; the depth of the parse tree; and the leftmost position of the word covered by the parse.

17.4.2. Statistical Pattern Matching

The use of CFGs to capture the semantic meaning of an utterance can be augmented with probabilistic CFGs or the unified language model described in Chapter 11. In the statistical parser, the application developers first define semantic nonterminal and preterminal nodes. A large number of sentences are then collected and annotated with these semantic nodes. The statistical training methods are used to build the parser to extract semantic meaning from an utterance.

For example, a statistical parsing algorithm [15, 26] takes one step further toward automatic discovery of complex CFG rules. Instead of relying on hand-written CFG rules, it builds a statistical parser based on the *tree-banked* data where sentences are labeled with parsing-tree structure. It identifies simple named classes like *Date, Amount, Fund*, or *Percent* and only handles simple classes using the local context. Words that are not part of a class are tagged as *word*, indicating that the word is passed on to the subsequent parser. The subsequent statistical parser takes a classed sentence. It generates the most likely semantic parse in a bottom-up leftmost derivation order. At each step in the derivation, the parsers use

Amazon/VB Assets
Exhibit 1012
Page 904

CART (see Chapter 4) to assign probabilities to primitive parser actions such as assigning a tag to a word or deciding when to begin a new constituent. A beam search is used to find the parse with highest probability. The two-step parsing for the sentence "*Please transfer one hundred dollars from voyager fund to fidelity fund*" is illustrated in Figure 17.16.

The *hidden understanding model* (HUM) [29, 30] is another statistical pattern matching techniques. Let $\mathbf{W}$ denote the sequence of words and S denote the meaning of the utterance. According to Bayes' rule, we have the following equation:

$$P(S \mid \mathbf{W}) = \frac{P(\mathbf{W} \mid S)P(S)}{P(\mathbf{W})} \tag{17.2}$$

The task of sentence interpretation can then be translated into finding the meaning representation $\hat{S}$, such that

$$\hat{S} = \arg\max_{S} P(\mathbf{W} \mid S)P(S) \tag{17.3}$$

$P(S)$ is the *semantic language model* that specifies the prior statistical distribution of meaning expressions. The semantic language model is based on a tree-structured meaning representation where concepts are represented as nodes in a semantic tree with subconcepts represented as child nodes. Figure 17.17 illustrates such a tree-structured meaning representation for the sentence "*United flight 203 from Dallas to Atlanta.*" The Flight concept has Airline, Flight_Ind, Flt_Num, Origin, and Destination subconcepts. Origin and Destination subconcepts have terminal nodes Origin_Ind and City and Dest_Ind and City, respectively. Each terminal node (like City) could be composed of a word or of a sequence of words.

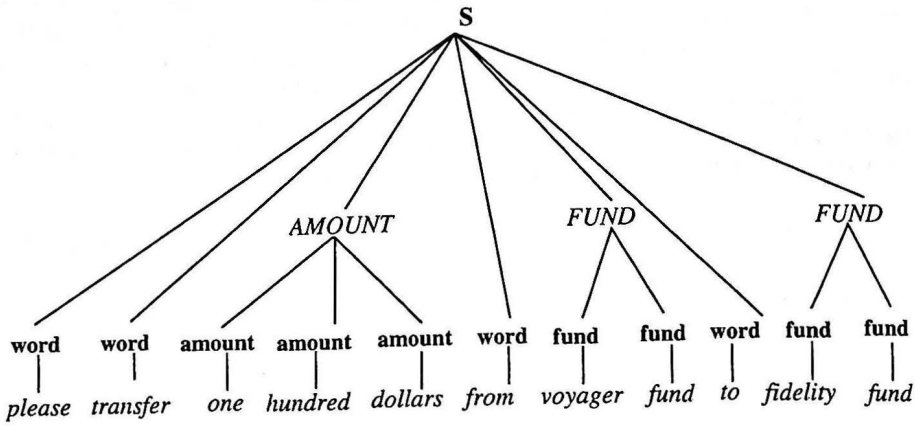
Semantic language model $P(S)$ is modeled as $P(S_i \mid S_{i-1}, concept)$, where *concept* is the parent concept for S_i and S_{i-1}. Based on this definition, the probability $P(\text{Destination} \mid \text{Origin}, \text{Flight})$ is bigger than $P(\text{Origin} \mid \text{Destination}, \text{Flight})$, since users often omit the origin for a flight in an airline reservation system.

$P(\mathbf{W} \mid S)$ is called a *lexical realization* model, which is basically a word bigram model augmented with the context of the parent concept:

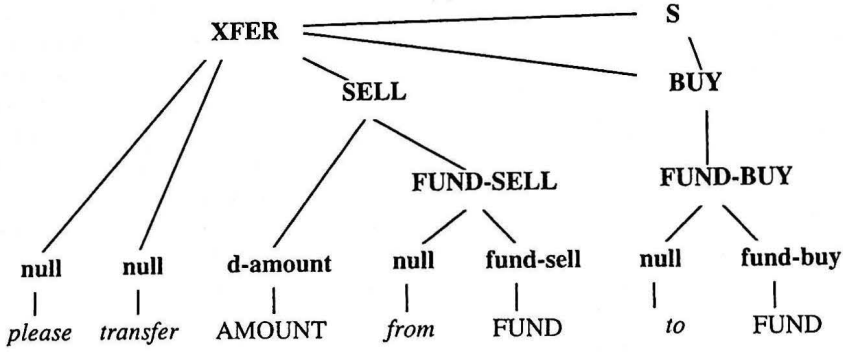
$$P(\mathbf{W} \mid S) = \prod P(w_i \mid w_{i-1}, concept) \tag{17.4}$$

Both the semantic language model and lexical realization model are estimated from a labeled corpus. Viterbi search is applied to find the best path of meaning representation $\hat{S}$ according to Eq. (17.3).

Amazon/VB Assets
Exhibit 1012
Page 905



(a)



(b)

Figure 17.16 An example class tree in IBM's statistical class parser. (a) The sentence is classified into semantic classes. (b) The classed sentence is parsed into the semantic tree based on CART [15].

```
FLIGHT  [AIRLINE[United]
          FLIGHT_IND[flight]
          FLIGHT_NUM[203]
          ORIGIN[ORIGIN_IND[from] CITY[DALLAS]]
          DESTINATION[DEST_IND[to] CITY[Atlanta]]]
```

Figure 17.17 A tree-structured meaning representation for *United flight 203 from Dallas to Atlanta* in BBN's HUM system [29].

Amazon/VB Assets
Exhibit 1012
Page 906

17.5. DISCOURSE ANALYSIS

The sentence interpretation module only attempts to interpret each sentence without knowledge about the current dialog status or discourse. As we mentioned in Section 17.2, sometimes it is impossible to get the right interpretation without discourse knowledge. For example, in the sentence "*Show me the morning flight*" one must have the knowledge what *the morning flight* refers to in order to derive the real-world entitiy, even though the sentence interpretation module comprehends perfectly what *morning flight* means.

Discourse information formed by dialog history is necessary not only for semantic inference but also for *inconsistency detection*. Inconsistency detection is important in a dialog system, since the *dialog management* module (described in Section 17.6) needs such information to disambiguate the dialog flow when needed. For example, in an airline reservation system, the returning date should not proceed the departure date, which may be conveyed in the previous dialog turns. The discourse analysis module needs to maintain a stack of discourse trees so that the semantic representation remains the same whether the information is obtained through several dialog turns or a single one.

The goal of the *discourse analysis* module is to collapse the discourse tree by resolving the semantic objects into the domain entities. This process is also called *semantic evaluation*. When the resolution is successful, the physical semantic object is officially bound to the domain entities. The last process is often called semantic binding. Because an entity can be identified by partial information (e.g., last name of a person), binding is necessary for the system to grasp the whole attributes of the objects the dialog is concerned with. Semantic binding is also critical for intelligent behaviors such as setting the discourse context for reference resolution. The semantic evaluation and binding are the basics for driving the dialog flow. The communication mechanism between discourse analysis and dialog manager is typically event driven. Events that can be passed to the dialog manager are *evaluation succeeded*, *evaluation failed*, *invalid information*, and *value to be determined*. The discourse analysis module often needs to tap into the knowledge base with the semantic object attributes and entity memory for semantic evaluation. The semantic evaluation usually proceeds from the leaves up toward the root of the discourse tree. The process ends when the root node is converted, which indicates the dialog goal has been achieved. The functions of *Discourse analysis* module are the following:

- Converting the *relative expressions* (like *tomorrow*, *next week*, *he*, *it*, *the morning flight*, etc.) in the semantic slots into real-world objects or concepts (such as 1/5/2000, *the week of 2/7/2000*, *John*, *John's dog*, etc.).
- Automatic inference—Based on dialog history, the module may decide some missing information for certain slots. For example, an airline reservation system could infer the destination city for the origin of the return flight even though it is not specified.
- Inconsistency/ambiguity detection—Since the discourse analysis module can perform automatic inference for some slots, it can perform consistency checking when it is explicitly specified during the current dialog turn.

Amazon/VB Assets
Exhibit 1012
Page 907